Министерство образования Республики Беларусь
Учреждение образования
«Гомельский государственный технический университет
имени П.О.Сухого»

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

КОНСПЕКТ ЛЕКЦИЙ
по дисциплине
«АЛГОРИТМИЧЕСКИЕ ОСНОВЫ СОВРЕМЕННОЙ КОМПЬЮТЕРНОЙ ГРАФИКИ»
для студентов специальности
1-40 05 01 «Информационные системы и технологии (по направлениям)»
направление специальности 1–40 05 01–01 «Информационные системы и технологии (в проектировании и производстве)»

Цитринов А.В.

Гомель 2016

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Дисциплина «Алгоритмические основы современной компьютерной графики» дает будущему специалисту знания и умения, позволяющие в дальнейшем эффективно использовать их в практической работе. Полученные знания и навыки будут также востребованы при изучении специальных дисциплин и станут инструментом в реализации программных продуктов в конкретной предметной области при выполнении курсовых и дипломных работ.

Дисциплина знакомит студентов с основами компьютерной графики и способами и алгоритмами отображения её на устройствах вывода.

Цель дисциплины – получение теоретических и практических знаний с использованием современных инструментов в области компьютерной графики, овладение практическими приемами работы с геометрическими объектами.

Задачи дисциплины – овладеть основами представления изображения в компьютерной графике. Изучить основы работы с двумерной и трёхмерной графикой, методы вывода изображения на экран. Приобрести навыки работы с алгоритмами отображения проекций трёхмерных объектов на плоскость, методами работы с цветом в задачах визуализации.

В результате изучения учебной дисциплины студент должен:

знать:

- области применения компьютерной графики;
- основные концепции и принципы отображения графической информации;
- технические средства поддержки компьютерной графики;
- программные средства поддержки компьютерной графики;
- цветовые модели, переход от одной модели к другой;
- алгоритмы геометрических преобразований, таких как параллельный перенос, масштабирование, вращение;

уметь:

- работать с программными средствами представления компьютерной графики;
- использовать технологии обработки графической информации;
- применять языки программирования для отображения геометрических объектов;

владеть:

- основами работы с двумерной и трёхмерной графикой;
- способами представления геометрической информации;
- алгоритмами отображения проекций трёхмерных объектов на плоскость;
- методами работы с цветом в задачах визуализации;
- принципами разработки программ для представления геометрической информации различными способами.

Основными методами обучения, отвечающими целям изучения дисциплины, являются:

- элементы проблемного обучения, реализуемое на лекционных занятиях;

- элементы учебно-исследовательской деятельности, реализуемые на лабораторных занятиях и при самостоятельной работе;
- коммуникативные технологии, реализуемые на лабораторных занятиях и конференциях.

Согласно учебному плану дисциплины «Алгоритмические основы современной компьютерной графики» на изучение дисциплины для очной формы получения образования отведено всего 106 часов, в том числе всего аудиторных часов – 51, из которых лекций – 34 часа, лабораторных занятия – 17 часов. В настоящем конспекте лекций рассмотрены только основные аспекты дисциплины «Алгоритмические основы современной компьютерной графики», такие как: основные направления применения компьютерной графики, история развития устройств ввода и вывода графической информации, форматы графических данных, понятия цвета и света в компьютерной графике, способы представления компьютерной графики, основные алгоритмы работы с графическими объектами. Несомненно, круг проблем и задач, решаемых в рамках дисциплины «Алгоритмические основы современной компьютерной графики» гораздо шире представленного в конспекте лекций материала, поэтому учебным планом предусмотрена самостоятельная работа студентов с литературой, обширный список которой приведен в конце конспекта лекций. Успешное освоение курса «Алгоритмические основы современной компьютерной графики» предполагает наличие у студентов базовых знаний из таких разделов высшей математики как линейная алгебра, алгебра матриц, аналитическая геометрия, а так же раздела «Оптика» из курса общей физики. Кроме того, в рамках курса предполагается наличие у студентов базовых практических навыков программирования и умения работы с блок-схемами алгоритмов.

Раздел 1. Понятие компьютерной графики. Понятие цвета и цветовых моделей

Глава 1. Общее введение в компьютерную графику

1 Предмет и область применения компьютерной графики

Компьютерная графика — это область информатики, которая охватывает все стороны формирования изображений с помощью компьютера. Появившись в 1950-х годах, она поначалу давала возможность выводить лишь несколько десятков отрезков на экране. В наши дни средства компьютерной графики позволяют создавать реалистические изображения, не уступающие фотографическим снимкам. Создано разнообразное аппаратное и программное обеспечение для получения изображений самого различного вида и назначения — от простых чертежей до реалистических образов естественных объектов. Компьютерная графика используется практически во всех научных и инженерных дисциплинах для наглядности восприятия и передачи информации. Применение её для подготовки демонстрационных слайдов уже считается нормой. Трёхмерные изображения используются в медицине (компьютерная томография[1]), картографии[2], полиграфии[3], геофизике[4], ядерной физике[5] и других областях. Телевидение и другие отрасли индустрии развлечений используют анимационные средства компьютерной графики (компьютерные игры, фильмы). Общепринятой практикой считается также использование компьютерного моделирования при обучении пилотов и представителей других профессий (тренажёры). Знание основ компьютерной графики сейчас необходимо и инженеру, и учёному.

Конечным результатом применения средств компьютерной графики является изображение, которое может использоваться для различных целей. Поскольку

---

[1] Томогра́фия (др.-греч. τομή — сечение) — получение послойного изображения внутренней структуры объекта.

[2] Картогра́фия (от греч. χάρτης — бумага из папируса + γράφειν — рисовать) — наука об исследовании, моделировании и отображении пространственного расположения, сочетания и взаимосвязи объектов и явлений природы и общества. В более широкой трактовке картография включает технологию и производственную деятельность.

[3] Полигра́фия (от др.-греч. πολύς — «многочисленный» и γράφω — «пишу») — это отрасль промышленности, занимающаяся размножением печатной продукции, а именно книжно-журнальной, деловой, газетной, этикеточной и упаковочной продукции.

[4] Геофизика (от др.-греч. γῆ — Земля + φύσις — природа) — комплекс наук, исследующих строение Земли физическими методами.

[5] Я́дерная фи́зика — раздел физики, изучающий структуру и свойства атомных ядер, а также их столкновения (ядерные реакции).

наибольшее количество информации человек получает с помощью зрения, уже в древние времена появились схемы и карты, используемые при строительстве, в географии и в астрономии.

Современная компьютерная графика — это достаточно сложная, основательно проработанная и разнообразная научно-техническая дисциплина. Некоторые её разделы, такие как геометрические преобразования, способы описания кривых и поверхностей, к настоящему времени уже исследованы достаточно полно. Ряд областей продолжает активно развиваться: методы растрового[6] сканирования, удаление невидимых линий и поверхностей, моделирование цвета и освещенности, текстурирование[7], создание эффекта прозрачности и полупрозрачности и др.

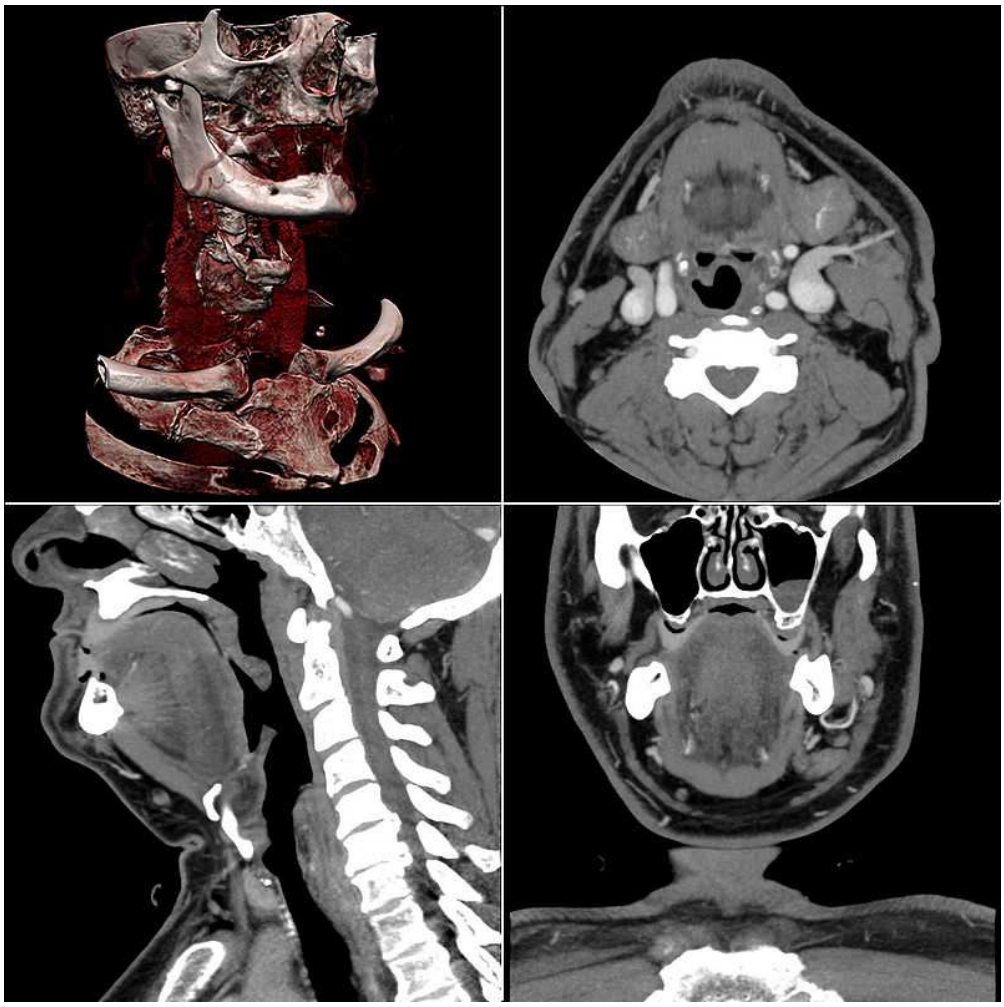


Рисунок 1 — Многослойная («мультиспиральная») компьютерная томография с внутривенным контрастным усилением и трёхмерной реконструкцией изображения.

[6] Растр (нем. Raster, от лат. Rastrum) — грабли. Растровое изображение — изображение, представляющее собой сетку пикселей или цветных точек (обычно прямоугольных) на мониторе, бумаге и других отображающих устройствах.

[7] Текстýра — изображение, воспроизводящее визуальные свойства каких-либо поверхностей или объектов.

Сфера применения компьютерной графики включает четыре основных области.

1.1 Отображение информации

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

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

В медицине в настоящее время широко используются методы диагностики, использующие компьютерную визуализацию внутренних органов человека. Томография (в частности, ультразвуковое исследование) позволяет получить трёхмерную информацию, которая затем подвергается математической обработке и выводится на экран. Помимо этого применяется и двумерная графика: энцефалограммы[8], миограммы[9], выводимые на экран компьютера или графопостроитель.

1.2 Проектирование

В строительстве и технике чертежи давно представляют собой основу проектирования новых сооружений или изделий. Процесс проектирования с необходимостью является итеративным[10], т.е. конструктор перебирает множество вариантов с целью выбора оптимального по каким-либо параметрам. Не последнюю

---

[8] Энцефалогра́фия (от др.-греч. ενκεφαλος — «головной мозг» + γραφω — «пишу») — методы измерения активности головного мозга. Энцефалогра́мма — графическое изображение, получаемое при исследовании головного мозга путём энцефалографии.

[9] Миогра́мма (электромиограмма, ЭМГ; от лат. myogram) — запись электрических сигналов, полученных в результате регистрации мышечных сокращений.

[10] Итерация (лат. iteratio — повторяю) — повторение какого-либо действия.

роль в этом играют требования заказчика, который не всегда чётко 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 расчёты и т.п. Они дают возможность не только изобразить проекции изделия, но и рассмотреть его в объёмном виде с различных сторон. Такие средства также чрезвычайно полезны для дизайнеров интерьера, ландшафта.

1.3 Моделирование

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

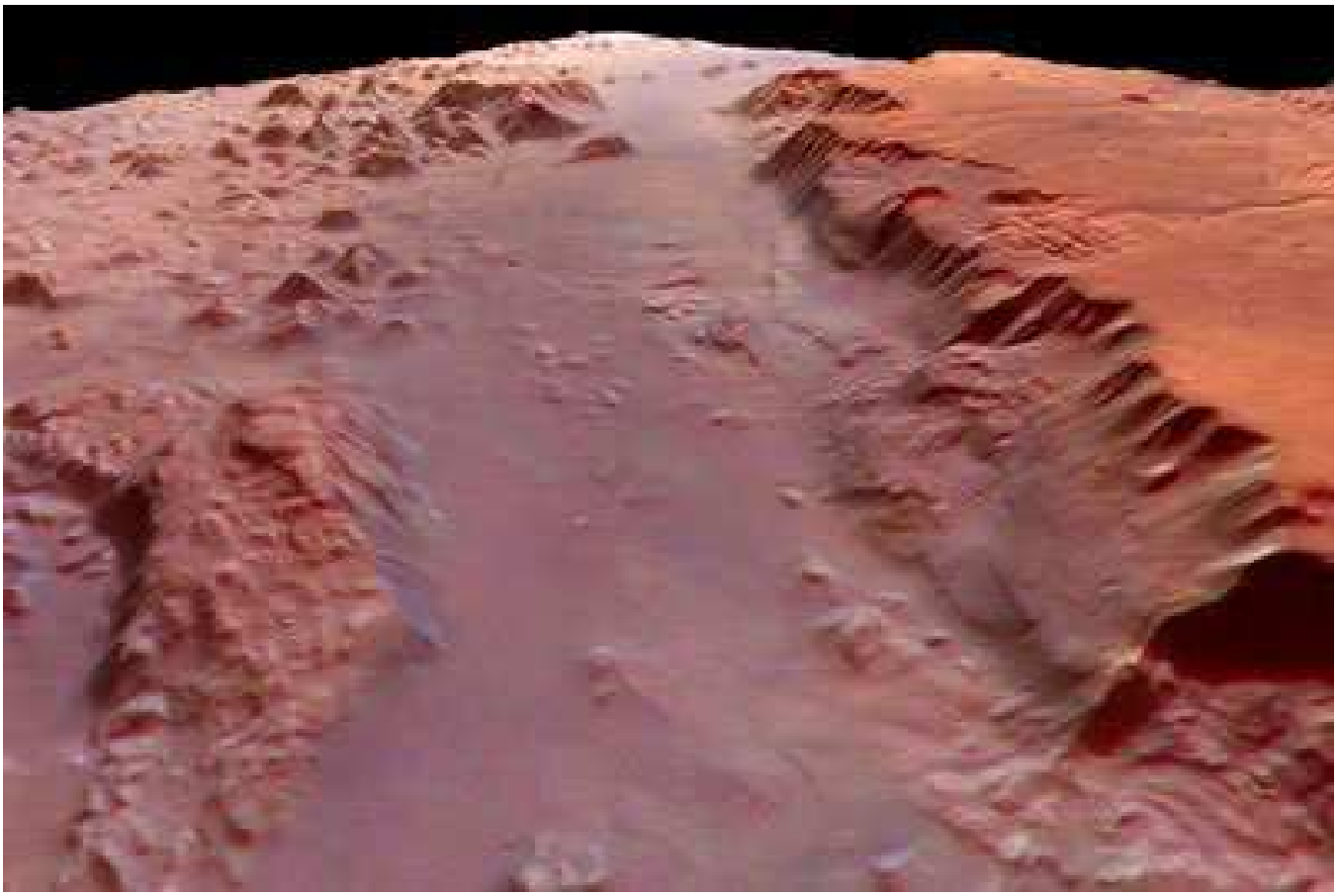


Рисунок 2 — Вид в перспективе был сгенерирован в соответствии с компьютерной моделью ландшафта планеты Марс, область Эос Часма (Eos Chasma).

Рисунок 3 — Компьютерная графика, используемая в художественном фильме «Звёздные войны» (1977г).

Рисунок 4 — Компьютерная графика, используемая в художественном фильме «Матрица» (1999г).

1.4 Графический пользовательский интерфейс

На раннем этапе использования дисплеев как одного из устройств компьютерного вывода информации диалог "человек-компьютер" в основном осуществлялся в алфавитно-цифровом виде (рисунок 5). Теперь же практически все системы программирования применяют графический интерфейс. Особенно впечатляюще выглядят разработки в области сети Internet. Существует множество различных программ-браузеров, реализующих в том или ином виде средства общения в сети, без которых доступ к ней трудно себе представить. Эти программы работают в различных операционных средах, но реализуют, по существу, одни и теже функции, включающие окна, баннеры, анимацию и т.д. (рисунок 6).

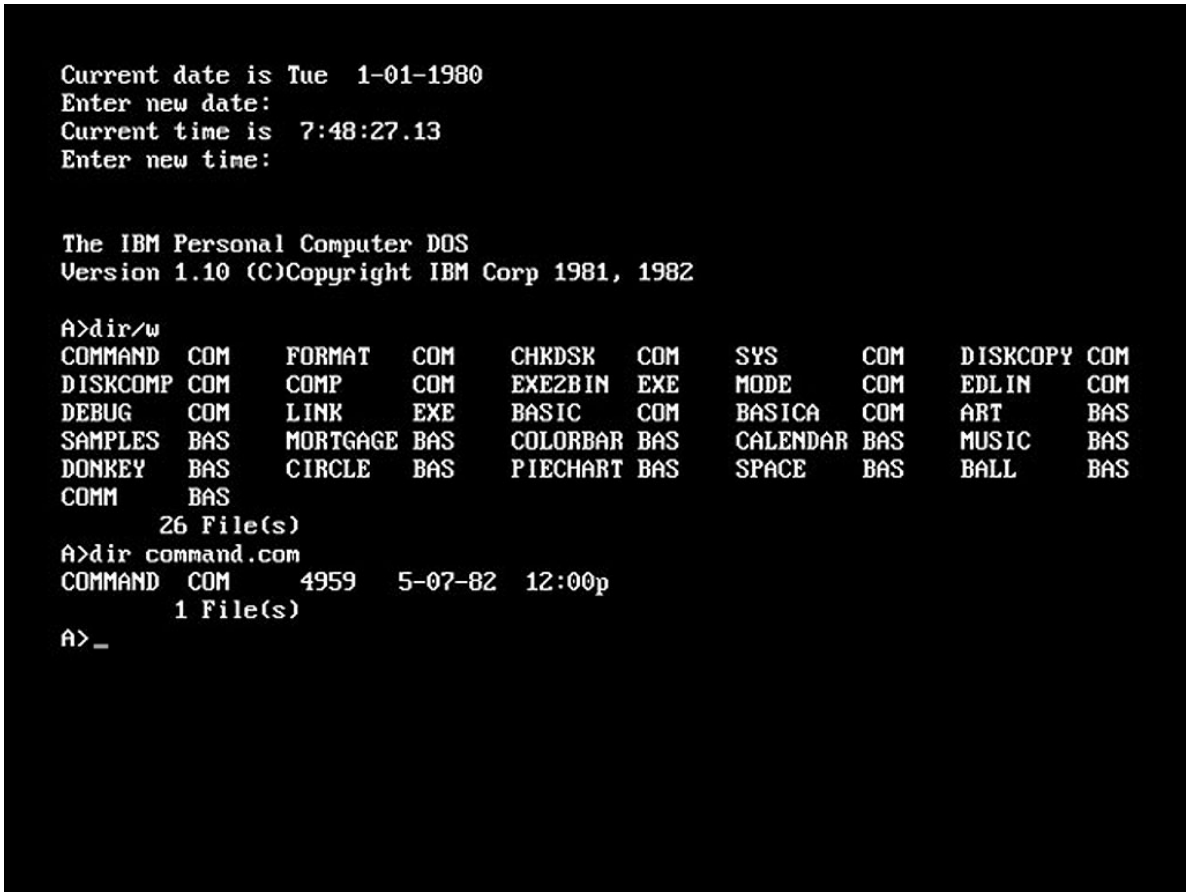


Рисунок 5 — Интерфейс ОС MS DOS.

Рисунок 6 — Графический интерфейс ОС MS Windows.

2 Основные направления современной компьютерной графики

В современной компьютерной графике можно выделить следующие основные направления: изобразительная компьютерная графика, обработка и анализ изображений, анализ сцен (перцептивная[11] компьютерная графика), компьютерная графика для научных абстракций (когнитивная [12] компьютерная графика, т.е. графика, способствующая познанию).

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

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

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

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

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

Когнитивная компьютерная графика — только формирующееся новое направление, пока ещё недостаточно чётко очерченное. Это — компьютерная графика для научных абстракций, способствующая рождению нового научного

---

[11] Перце́пция (от лат. perceptio — восприятие) — познавательный процесс, формирующий субъективную картину мира.

[12] Когнитивный (от лат. cognitio — познание) — познавательный.

знания. Технической основой для неё являются мощные ЭВМ и высокопроизводительные средства визуализации.

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

В настоящем курсе предполагается рассмотреть следующие вопросы:

1. представление изображения в компьютерной графике;
2. способы подготовки изображения к визуализации;
3. методы вывода изображения на экран;
4. методы работы с изображением;
5. методы вычислительной геометрии.

3 Технические средства поддержки компьютерной графики

3.1 Устройства вывода

Развитие компьютерной графики во многом обусловлено развитием технических средств её поддержки. Прежде всего, это устройства вывода, каковыми являются дисплеи. В настоящее время существует несколько типов дисплеев, использующих электронно-лучевую трубку, а также дисплеи на жидкокристаллических индикаторах и другие их виды. Рассмотрим функциональные возможности дисплеев, не касаясь их внутреннего устройства и электронных схем.

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

Самым простым из устройств на ЭЛТ является дисплей на запоминающей трубке с прямым копированием изображения. Запоминающая трубка обладает свойством длительного времени послесвечения: изображение остается видимым в течение длительного времени (до одного часа). При выводе изображения интенсивность электронного луча увеличивают до уровня, при котором происходит запоминание следа луча на люминофоре[13]. Сложность изображения практически не ограничена. Стирание происходит путём подачи на всю трубку специального напряжения, при котором свечение исчезает, и эта процедура занимает приблизительно 0,5 с. Поэтому изображения, полученные на экране, нельзя стереть частично, а стало быть, динамические изображения или анимация на таком дисплее невозможны. Дисплей на запоминающей трубке является векторным, или дисплеем с произвольным сканированием, т.е. он позволяет провести отрезок из одной адресуемой точки в любую другую. Его достаточно легко программировать, но уровень интерактивности у него ниже, чем у ряда дисплеев других типов ввиду низкой скорости и плохих характеристик стирания.

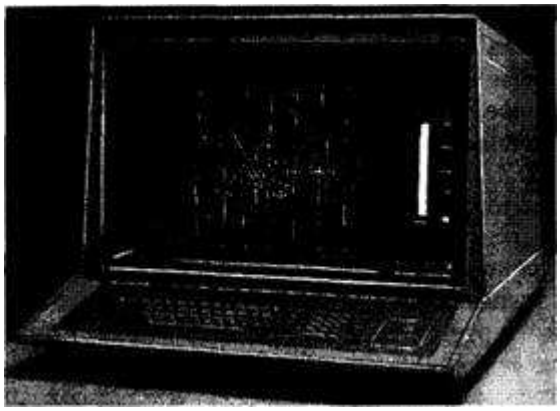


Рисунок 7 — Графический дисплей на запоминающей трубке.

Следующий тип — это *векторные дисплеи с регенерацией изображения*. При перемещении луча по экрану в точке, на которую попал луч, возбуждается свечение люминофора экрана. Это свечение достаточно быстро прекращается при перемещении луча в другую позицию (обычное время послесвечения — менее 0,1 с). Поэтому, для того чтобы изображение было постоянно видимым, приходится его "перерисовывать" (регенерировать изображение) 50 или 25 раз в секунду. Необходимость регенерации изображения требует сохранения его описания в специально выделенной памяти, называемой памятью регенерации. Само описание изображения называется дисплейным файлом. Понятно, что такой дисплей требует

[13] Люминофо́р (от лат. lumen — свет и др.-греч. φορός — несущий) — вещество, способное преобразовывать поглощаемую им энергию в световое излучение (люминесцировать).

достаточно быстрого процессора для обработки дисплейного файла и управления перемещением луча по экрану.

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

Рисунок 8 — Устройство отображения на радиолокационных станциях и в авиадиспетчерских.

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

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

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

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

Прогресс в технологии микроэлектроники привел к тому, что с середины 1970-х годов преимущественное распространение получили дисплеи с растровым сканированием луча. Растровое устройство можно рассматривать как матрицу дискретных[14] точек (пикселей), каждая из которых может быть подсвечена. Таким образом, оно является точечно рисующим устройством. Поэтому любой изображаемый на экране дисплея отрезок строится с помощью последовательности точек, аппроксимирующих[15] идеальную траекторию отрезка, подобно тому, как можно строить изображение по клеткам на клетчатом листке бумаги. При этом отрезок получается прямым только в случаях, когда он горизонтален, вертикален или направлен под углом 45° к горизонтали. Все другие отрезки выглядят как последовательность "ступенек" (ступенчатый эффект, рисунок 9).

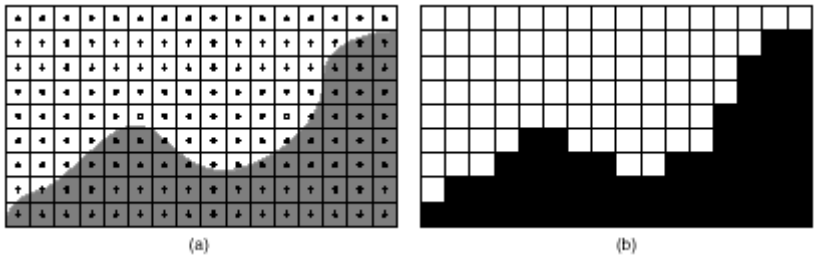


Рисунок 9 — Ступенчатый эффект.

При построении изображения в растровых графических устройствах используется буфер[16] кадра, представляющий собой большой непрерывный участок памяти компьютера. Для каждой точки в растре отводится, как минимум, один бит памяти. Буфер кадра сам по себе не является устройством вывода, он лишь используется для хранения рисунка. Наиболее часто в качестве устройства вывода, используемого с буфером кадра, выступает видеомонитор.

Чтобы понять принципы работы растровых дисплеев, можно рассмотреть в общих чертах устройство цветной растровой электронно-лучевой трубки.

---

[14] Дискре́тность (от лат. discretus — разделённый, прерывистый) — свойство, противопоставляемое непрерывности, прерывность.

[15] Аппроксима́ция, или приближе́ние — научный метод, состоящий в замене одних объектов другими, в каком-то смысле близкими к исходным, но более простыми.

[16] Бу́фер (применительно к информатике) — область памяти, используемая для временного хранения данных ввода-вывода.

Изображение на экране получается с помощью сфокусированного электронного луча, который, попадая на экран, покрытый люминофором, дает яркое цветовое пятно. Луч в растровом дисплее может отклоняться только в строго определённые позиции на экране, образующие своеобразную мозаику. Люминофорное покрытие тоже не непрерывно, а представляет собой множество близко расположенных мельчайших точек, куда может позиционироваться луч. Дисплей, формирующий чёрно-белые изображения, имеет одну электронную пушку, и её луч высвечивает однотонные цветовые пятна. В цветной ЭЛТ находятся три электронных пушки, по одной на каждый основной цвет: красный, зелёный и синий. Электронные пушки часто объединены в треугольный блок, соответствующий треугольным блокам красного, зелёного и синего люминофоров на экране. Электронные лучи от каждой из пушек, проходя через специальную теневую маску, попадают точно на пятно своего люминофора. Изменение интенсивности каждого из трёх лучей позволяет получить не только три основных цвета, но и цвета, получаемые при их смешении в разных пропорциях, что дает очень большое количество цветов для каждого пикселя экрана.

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

Существуют и другие виды дисплеев, например плазменная панель, и другие. В настоящее время в широком применении дисплеи являются растровыми. Важно то, что рассматриваемые алгоритмы разработаны для растровых графических дисплеев, а общие принципы работы этих устройств понятны.

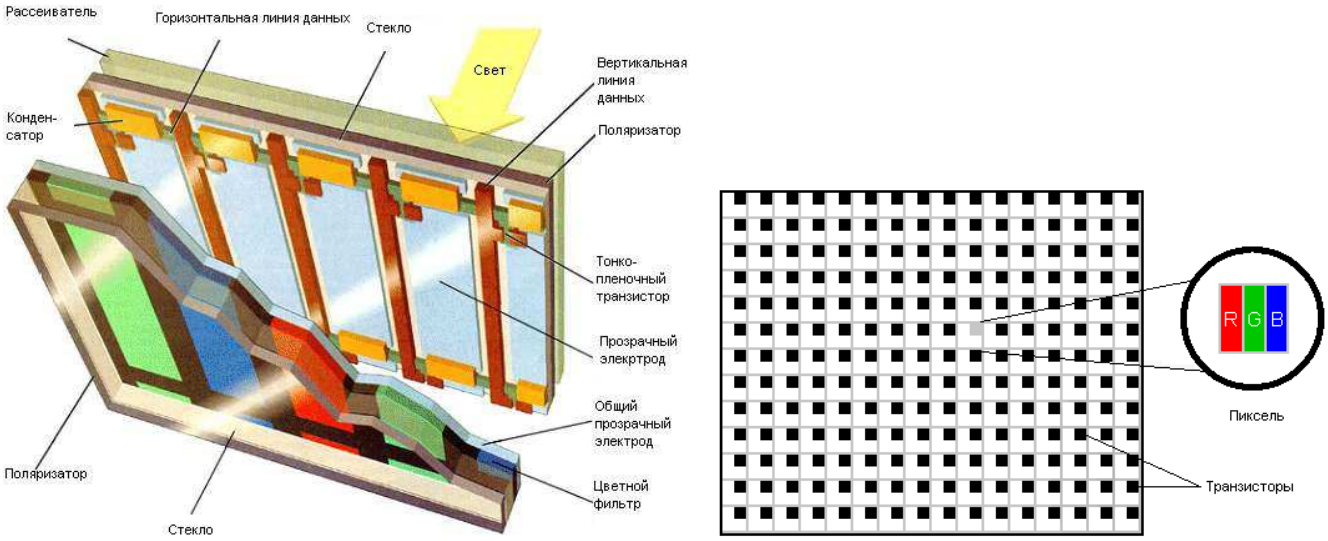


Рисунок 10 — Устройство жидкокристаллического дисплея.

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

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

3.2 Устройства ввода

Необходимо сделать небольшой обзор устройств ввода информации, позволяющих решать различные задачи компьютерной графики, не вдаваясь в детали физических принципов их работы. Эти устройства позволяют организовать диалог "человек-компьютер", а особенности конструкции каждого устройства

позволяют ему специализироваться на выполнении определённого круга задач. В данном случае эти устройства интересны именно как логические, т.е. с точки зрения выполняемых ими функций.

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

Мышь (рисунок 11) и *трекбол* (рисунок 12) похожи не только по назначению, но часто и по конструкции. В механической мыши и трекболе вращение шарика преобразуется с помощью пары преобразователей в сигналы, передаваемые в компьютер. Преобразователи измеряют вращение относительно двух взаимно перпендикулярных осей. Существует очень много модификаций устройств этих групп. В оптической мыши используются не механические, а оптические чувствительные элементы для измерения перемещения: измеряется расстояние путём подсчета штрихов на специальной подложке. Маленькие трекболы широко применяются в портативных компьютерах, где их встраивают прямо в клавиатуру.

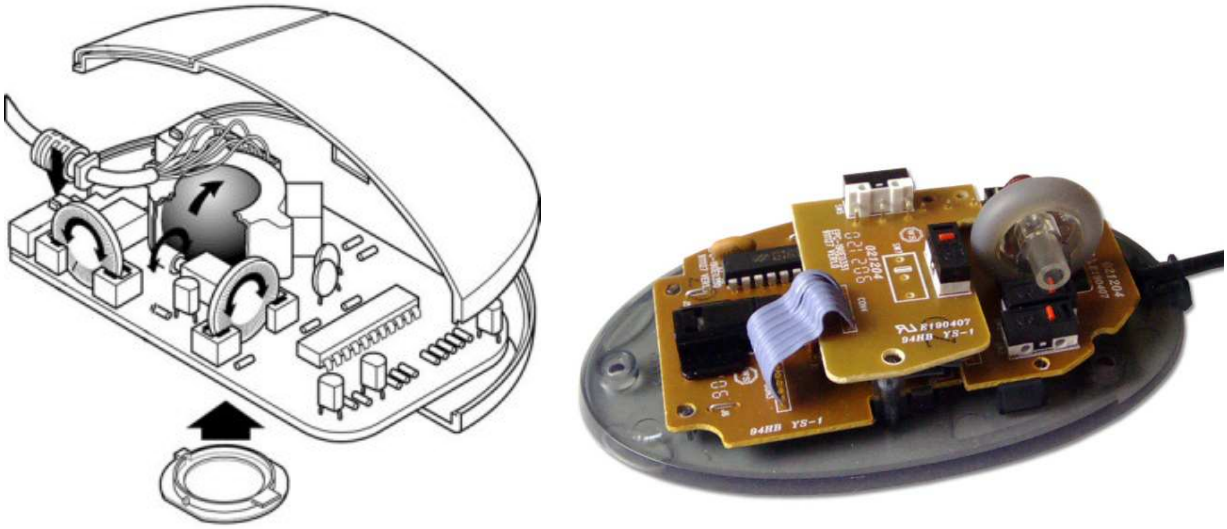


Рисунок 11 — Устройство манипулятора «мышь».

Рисунок 12 — Трекбол.

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

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

Ветераном среди устройств ввода в компьютерной графике является устройство, названное при его создании *световым пером* (рисунок 13). Световое перо содержит фоточувствительный элемент, который при приближении к экрану воспринимает излучение, порождаемое при столкновении электронов с люминофорным покрытием экрана. Если мощность светового импульса превышает определённый порог, фоточувствительный элемент формирует импульс, который передается в компьютер. Анализируя смещение по времени этого импульса относительно начала цикла регенерации, компьютер может точно определить координаты той точки экрана, возбуждение которой "высветило" фотоэлемент. Таким образом, в распоряжении пользователя оказывается устройство непосредственного указания, работающее напрямую с изображением на экране. В настоящее время это устройство уже практически вышло из употребления: оно вытеснено более простым и надежным — мышью.

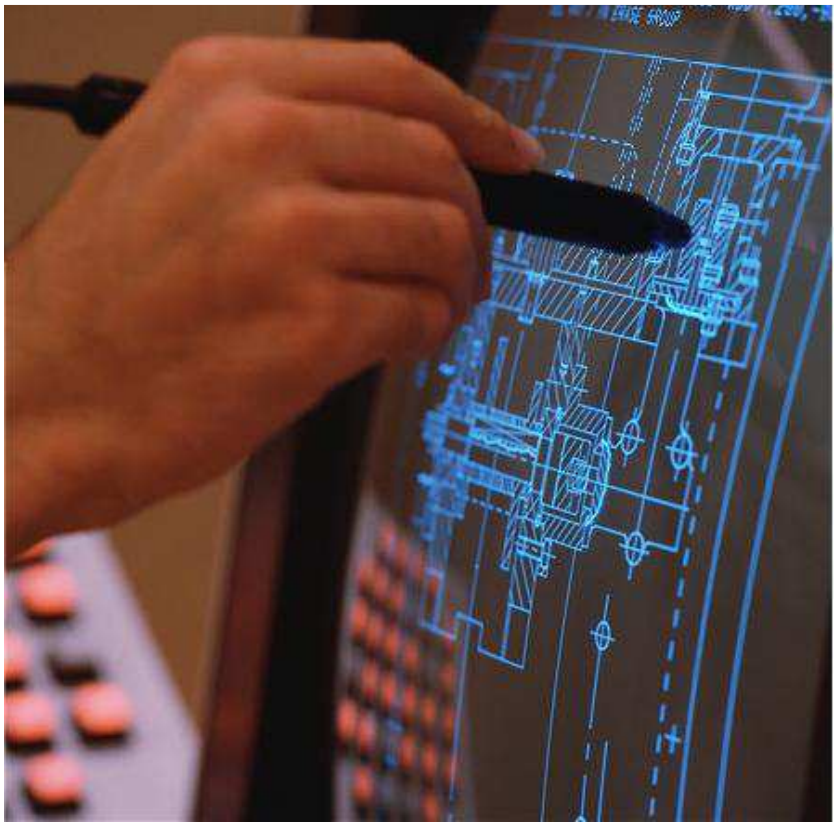


Рисунок 13 — Световое перо.

Ещё одно устройство, которое достаточно активно используется в мультимедийных приложениях, а также в различного рода компьютерных тренажёрах — джойстик (рисунок 14). Перемещение джойстика в двух взаимно перпендикулярных направлениях воспринимается преобразователями, интерпретируется как вектор скорости, а полученные значения используются для управления положением маркера на экране. Обработка сигнала выполняется таким образом, что неподвижный джойстик в каком-либо промежуточном положении не изменяет положения маркера, а чем дальше джойстик отклонен от начального положения, тем быстрее маркер перемещается по экрану. Таким образом, джойстик играет роль устройства ввода с переменной чувствительностью. Другое достоинство джойстика — наличие силовой обратной связи, обеспеченной наличием разного рода пружин. При этом пользователь чувствует, что чем дальше отклонен джойстик, тем большее усилие требуется для его дальнейшего движения. Это как раз те свойства, которые нужны при работе с разного рода симуляторами, а также в компьютерных играх.

Рисунок 14 — Джойстик.

Спейсбол — это "трёхмерное" устройство ввода. Хотя и существуют различные конструкции таких устройств, они всё ещё не получили широкого распространения, поскольку проигрывают популярным двумерным устройствам как по стоимости, так и по техническим характеристикам. Спейсбол похож на джойстик, но отличается от него тем, что он имеет вид закрепленного на рукоятке шара, причём рукоятка в этой конструкции неподвижна (рисунок 15). Шар имеет датчики давления, которые измеряют усилие, прикладываемое пользователем. Шар может измерять не только составляющие усилия в трёх основных направлениях (сверху вниз, от себя или на себя, влево-вправо), но и вращение относительно трёх осей. Таким образом, это устройство способно передавать в компьютер шесть независимых параметров (т. е. имеет шесть степеней свободы), характеризующих как поступательное движение, так и вращение.

Рисунок 15 — Спейсбол.

Существуют и другие трёхмерные системы измерения и ввода, использующие самые современные технологии, например лазерные. В системах виртуальной реальности используются более сложные устройства, позволяющие динамически отслеживать положение и ориентацию пользователя. Для приложений, связанных с современной робототехникой и моделированием виртуальной реальности, иногда требуются устройства, обладающие ещё большим числом степеней свободы, чем спейсбол. В последнее время появились новые разработки в этом направлении, в частности — перчатки с системой датчиков, которые способны улавливать движения отдельных частей руки человека (рисунок 16).

Рисунок 16 — Сенсорные перчатки.

При использовании мыши или трекбола анализируется относительное положение устройства. Если переместить указатель на экране каким-либо способом в другое место, не вращая при этом шарик мыши или трекбола, то дальнейшие сигналы будут смещать указатель относительно новой позиции. Можно также аккуратно переместить мышь без вращения шарика и это не приведет к перемещению курсора на экране. Абсолютные координаты устройства не считываются обрабатывающей программой. Но при вводе в компьютер графиков прикладной программе зачастую требуются абсолютные координаты устройства ввода. Такую возможность обеспечивают разного рода планшеты (рисунок 17). В планшете применяется, как правило, ортогональная сетка проводов, расположенная под его поверхностью. Положение пера определяется через электромагнитное взаимодействие сигналов, проходящих от проводов к щупу. Иногда в качестве планшета используются чувствительные к прикосновению прозрачные экраны, которые наносятся на поверхность ЭЛТ. Небольшие экраны такого типа размещаются иногда на клавиатуре портативных компьютеров. Чувствительные

панели можно использовать в режимах как абсолютных, так и относительных координат.

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

Рисунок 17 — Планшет.

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

4 Программные средства поддержки компьютерной графики

Для обработки изображений на компьютере используются специальные программы — графические редакторы. Графический редактор — это программа создания, редактирования и просмотра графических изображений. Графические редакторы можно разделить на две категории: растровые и векторные.

4.1 Растровые графические редакторы

Растровые графические редакторы являются наилучшим средством обработки фотографий и рисунков, поскольку растровые изображения обеспечивают высокую точность передачи градаций цветов и полутонов. Среди растровых графических редакторов есть простые, например, стандартное приложение Paint, и мощные профессиональные графические системы, например, Adobe Photoshop, или более упрощённый аналог — Paint.NET.

Растровое изображение хранится с помощью точек различного цвета (пикселей [17]), которые образуют строки и столбцы. Любой пиксель имеет фиксированное положение и цвет. Хранение каждого пикселя требует некоторого количества бит информации, которое зависит от количества цветов в изображении.

Качество растрового изображения определяется размером изображения (числом пикселей по горизонтали и вертикали) и количества цветов, которые могут принимать пиксели.

Растровые изображения очень чувствительны к масштабированию (увеличению или уменьшению). Когда растровое изображение уменьшается, несколько соседних точек превращаются в одну, поэтому теряется разборчивость мелких деталей изображения. При укрупнении изображения увеличивается размер каждой точки и появляется ступенчатый эффект, который виден невооруженным глазом (рисунок 18).

[17] Пи́ксель, пи́ксел (иногда пэл, англ. pixel, pel — сокращение от pix element, в некоторых источниках picture cell — букв. элемент изображений) или элиз (редко используемый русский вариант термина) — наименьший логический элемент двумерного цифрового изображения в растровой графике, или [физический] элемент матрицы дисплеев, формирующих изображение.

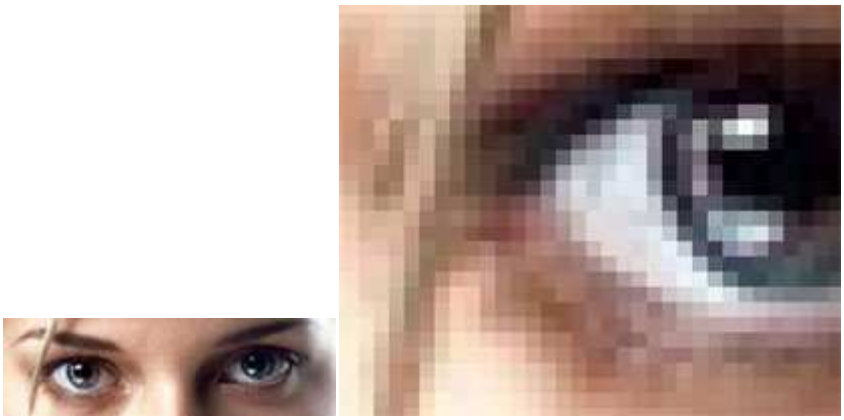


Рисунок 18 — Эффект масштабирования растрового изображения.

4.2 Векторные графические редакторы

Векторные графические изображения являются оптимальным средством для хранения высокоточных графических объектов (чертежи, схемы и т. д.), для которых имеет значение наличие чётких и ясных контуров. С векторной графикой сталкиваются, когда работают с системами компьютерного черчения и автоматизированного проектирования, с программами обработки трёхмерной графики.

К *векторным графическим редакторам* относятся графический редактор, встроенный в текстовый редактор MS Word. Среди профессиональных векторных графических систем наиболее распространены CorelDRAW и Adobe Illustrator.

Векторные изображения формируются из объектов (точка, линия, окружность и т. д.), которые хранятся в памяти компьютера в виде графических примитивов и описывающих их математических формул.

Например, графический примитив точка задается своими координатами (X, Y), линия — координатами начала (XI, У1) и конца (X2, Y2), окружность — координатами центра (X, Y) и радиусом (К), прямоугольник — величиной сторон и координатами левого верхнего угла (Xl, Y1) и правого нижнего угла (X2, Y2) и т. д. Для каждого примитива назначается также цвет.

Достоинством векторной графики является то, что файлы, хранящие векторные графические изображения, имеют сравнительно небольшой объём. Важно также, что векторные графические изображения могут быть увеличены или уменьшены без потери качества.

4.3 Форматы графических файлов

Форматы графических файлов определяют способ хранения информации в файле (растровый или векторный), а также форму хранения информации (используемый алгоритм сжатия).

Сжатие применяется для растровых графических файлов, так как они имеют обычно достаточно большой объём. Сжатие графических файлов отличается от их архивации с помощью программ архиваторов тем, что алгоритм сжатия включается в формат графического файла.

Некоторые форматы графических файлов являются универсальными, так как могут быть обработаны большинством графических редакторов. Некоторые программы обработки изображений используют оригинальные форматы, которые распознают только самой создающей программой.

Примеры некоторых форматов графических файлов приведены ниже:

- BMP — универсальный формат растровой графики.
- GIF — формат растровых графических файлов для различных ОС. Используется для размещения графических изображений в Интернете. Наиболее часто употребляется для анимированных изображений.
- JPEG — формат растровых графических файлов, который использует эффективных алгоритм сжатия (с потерями). Используется для размещения графических изображений в Интернете.
- WMF — универсальных формат векторных графических файлов для windows-приложений.
- CDR — оригинальный формат векторных графических файлов, используется в системе обработки изображений CorelDraw.

Глава 2. Природа света и цвета. Цветовой график МКО

1 О природе света и цвета

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

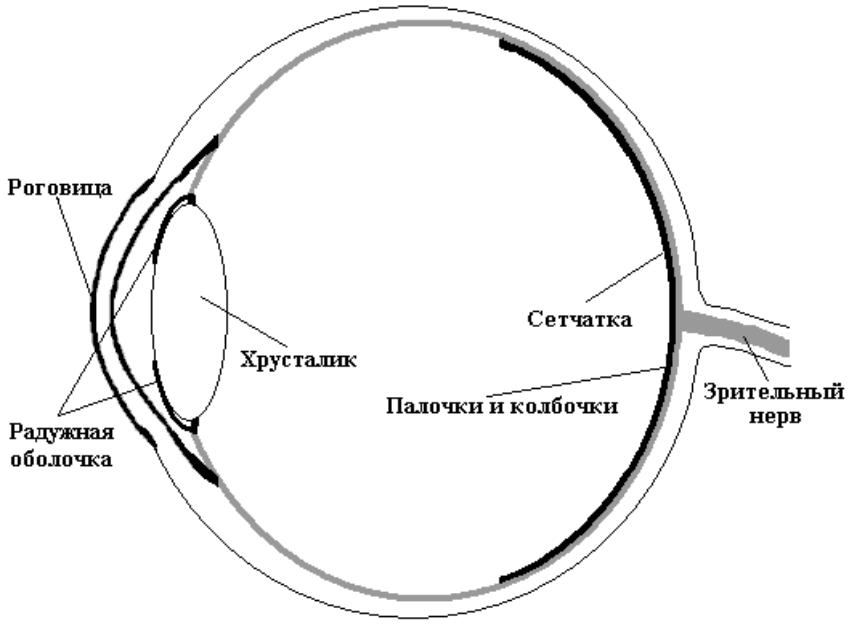


Рисунок 19 — Глаз человека.

На рисунке 19 схематически изображен глаз человека. Фоторецепторы, расположенные на поверхности сетчатки, играют роль приемников света. Хрусталик — это своеобразная линза, формирующая изображение, а радужная оболочка исполняет роль диафрагмы, регулируя количество света, пропускаемого внутрь глаза. Чувствительные клетки глаза неодинаково реагируют на волны различной длины. Интенсивность света есть мера энергии света, воздействующего на глаз, а яркость — это мера восприятия глазом этого воздействия. Интегральная

кривая спектральной чувствительности глаза приведена на рисунке 20; это стандартная кривая Международной комиссии по освещению (МКО, или CIE — Comission International de l'Eclairage).

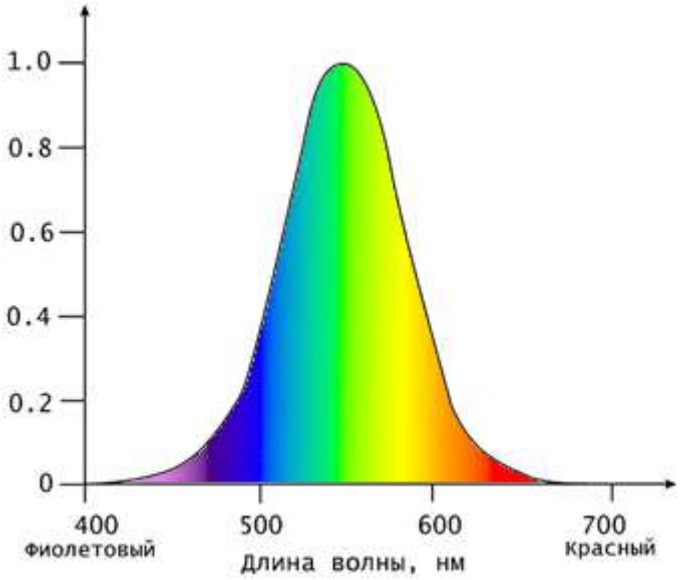


Рисунок 20 — Интегральная кривая спектральной чувствительности глаза.

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

На рисунке 21 представлены кривые чувствительности колбочек для всех трёх видов. Видно, что наибольшей чувствительностью обладают колбочки, воспринимающие цвета зелёного спектра, немного слабее — "красные" колбочки и существенно слабее — "синие".

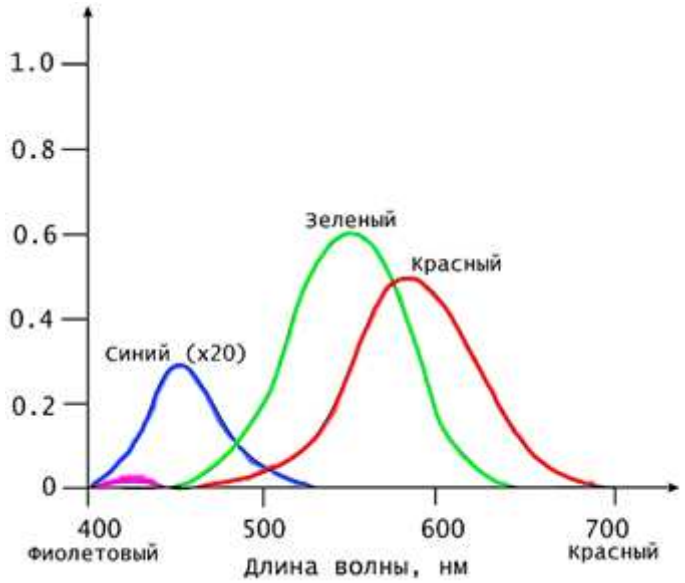


Рисунок 21 — Кривые чувствительности различных рецепторов.

Таким образом, если функция $C(\lambda)$ характеризует спектральное разложение светового излучения от некоторого источника (рисунок 22), т. е. распределение интенсивности по длинам волн, то три типа колбочек будут посылать в мозг сигналы R, G, B (R – red, красный; G – green, зелёный; B – blue, синий), мощность которых определяется интегральными соотношениями:

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

$$G = \int C(\lambda) S_G(\lambda)\, d\lambda, \qquad (1)$$

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

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

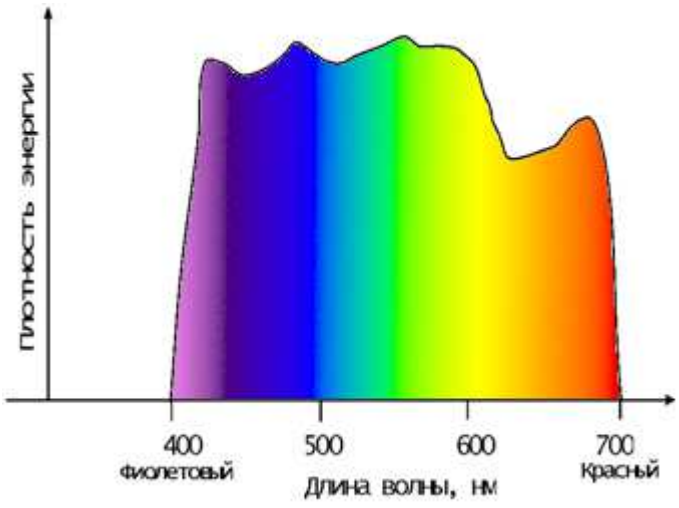


Рисунок 22 — Характерная спектральная кривая.

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

2 Цветовой график МКО

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

$$r = \frac{R}{R+G+B},$$

$$g = \frac{G}{R+G+B}, \qquad (2)$$

$$b = \frac{B}{R+G+B}.$$

Поскольку эти координаты в сумме всегда составляют единицу, а каждая из координат лежит в диапазоне от 0 до 1, то все представленные таким образом точки пространства будут лежать в одной плоскости, причём только в треугольнике, отсекаемом от неё положительным октантом системы координат (рисунок 23а). Ясно, что при таком представлении всё множество точек этого треугольника можно описать с помощью двух координат, так как третья выражается через них посредством соотношения:

$$b=1-r-g \qquad (3)$$

Таким образом, переходим к двумерному представлению области, т.е. к проекции области на плоскость (рисунок 23б).

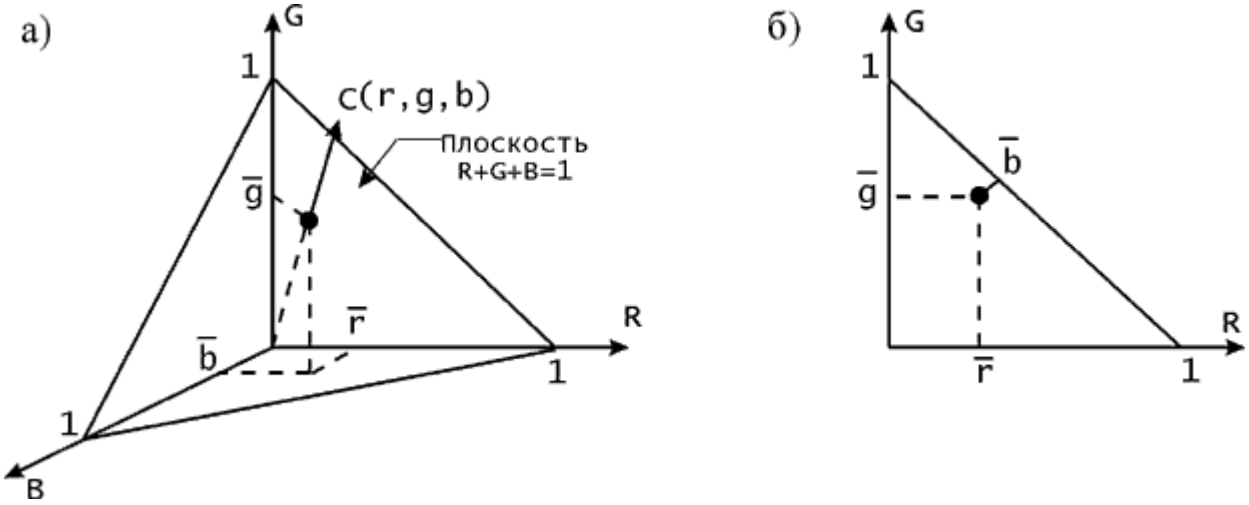


Рисунок 23 — Трёхмерное цветовое пространство.

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

$$x = \frac{X}{X+Y+Z},$$

$$y = \frac{Y}{X+Y+Z}, \qquad (4)$$

$$z = \frac{Z}{X+Y+Z},$$

$$x+y+z=1$$

При проекции этого треугольника на плоскость получается цветовой график МКО. Но координаты цветности определяют только относительные количества

основных цветов, не задавая яркости результирующего цвета. Яркость можно задать координатой Y, а X, Z определить исходя из величин (x,y,Y), по формулам:

$$X = \frac{Y}{y}x$$

$$Z = \frac{Y}{y}(1 - x - y) \tag{5}$$

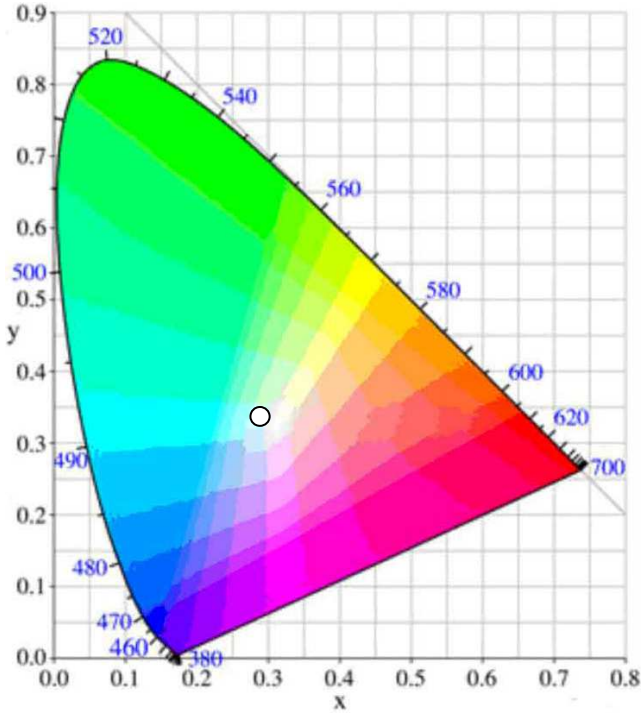


Рисунок 24 — Цветовой график МКО. На контуре указаны длины волн в нанометрах.

Цветовой график МКО приведён на рисунке 24. Область, ограниченная кривой, охватывает весь видимый спектр, а сама кривая называется линией спектральных цветностей. Числа, проставленные на рисунке 24, означают длину волны в соответствующей точке. Точка C, соответствующая полуденному освещению при сплошной облачности, принята в качестве опорного белого цвета.

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

Для смешения двух цветов используются *законы Грассмана*.

Пусть два цвета заданы на графике МКО координатами:

$D_1=(x_1, y_1, Y_1)$ и $D_2=(x_2, y_2, Y_2)$.

Тогда смешение даёт цвет: $D_{12}=(x_1+x_2, y_1+y_2, z_1+z_2)$.

Если ввести обозначения $t_1 = \frac{Y_1}{y_1}$, $t_2 = \frac{Y_2}{y_2}$, то получим координаты цветности смеси:

$x_{12} = \frac{x_1t_1+x_2t_2}{t_1+t_2}$, $y_{12} = \frac{y_1t_1+y_2t_2}{t_1+t_2}$, $Y_{12}=Y_1+Y_2$.

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

3 Цветовые модели RGB и CMY

Цветовые модели, используемые в компьютерной графике, — это средства описания цветов в определённом диапазоне.

Цветовая модель — математическая модель[18] описания представления цветов в виде кортежей[19] чисел (обычно из трёх, реже — четырёх значений), называемых цветовыми компонентами или цветовыми координатами. Все возможные значения цветов, задаваемые моделью, определяют цветовое пространство.

На основе описанных выше физических представлений в компьютерной графике была принята так называемая *аддитивная* (от англ. addition — добавить) цветовая модель, использующая три первичных составляющих цвета. Эта модель предполагает, что любой цвет можно рассматривать как взвешенную сумму трёх основных цветов. Проиллюстрировать её можно на примере освещения сцены с помощью трёх прожекторов разного цвета (рисунок 25). Каждый прожектор

[18] Математи́ческая моде́ль — математическое представление реальности, один из вариантов модели, как системы, исследование которой позволяет получать информацию о некоторой другой системе.

[19] Кортеж (от фр. cortège — шествие) — упорядоченный набор фиксированной длины.

управляется независимо, и путём изменения мощности каждого из них можно воспроизвести практически все цвета.

В модели *RGB* цвет можно представить в виде вектора в трёхмерной системе координат с началом отсчета в точке (0,0,0). Максимальное значение каждой из компонент вектора примем за 1. Тогда вектор (1,1,1) соответствует белому цвету. Все цветовые векторы, таким образом, заключены внутри единичного куба, называемого цветовым кубом (рисунок 26).

Рисунок 25 — Аддитивное смешение цветов.

Другая модель смешения цветов — *субстрактивная* (от англ. subtraction — вычитание) цветовая модель, или модель *CMY*, использующая в качестве первичных составляющих цвета Cyan, Magenta, Yellow (голубой (сине-зелёный), пурпурный, жёлтый), которые являются дополнительными к Red, Green, Blue. В этой модели оттенки цвета получаются путём "вычитания" из падающего света волн определённой длины. В этой системе координат вектор (0,0,0) соответствует белому цвету, а вектор (1,1,1) — чёрному. Соответствующий цветовой куб представлен на рисунке 27.

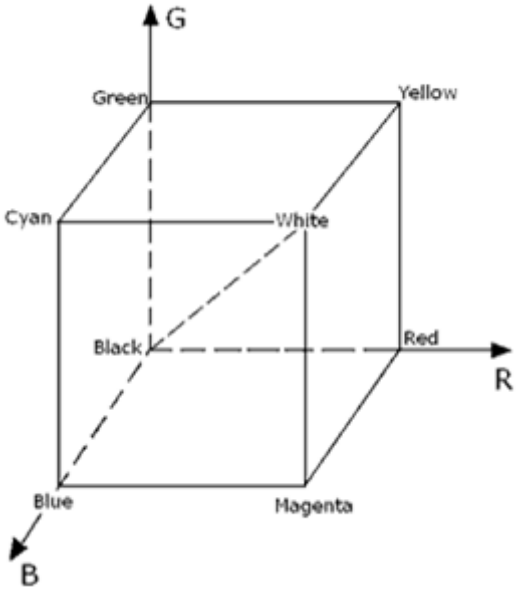


Рисунок 26 — Цветовой куб модели RGB.

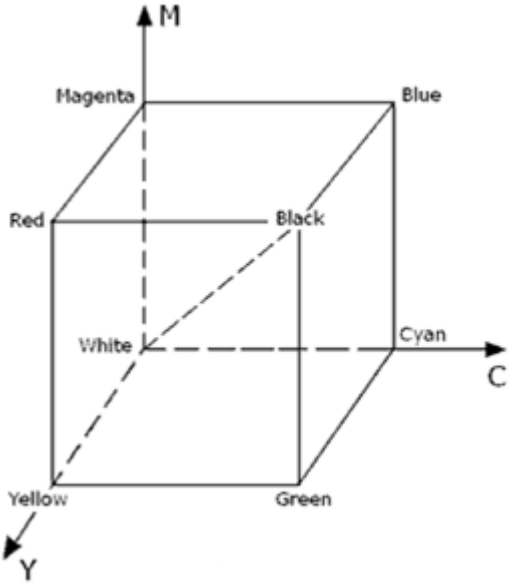


Рисунок 27 — Цветовой куб модели CMY.

Связь между значениями (R,G,B) и (C,M,Y) для одного и того же цвета выражается формулой (6):

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} C \\ M \\ Y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}. \quad (6)$$

Цвета одной модели являются дополнительными к цветам другой. Дополнительный цвет — это цвет, результатом смешения которого с данным является белый.

Схема смешения цветов для двух моделей представлена на рисунке 28.

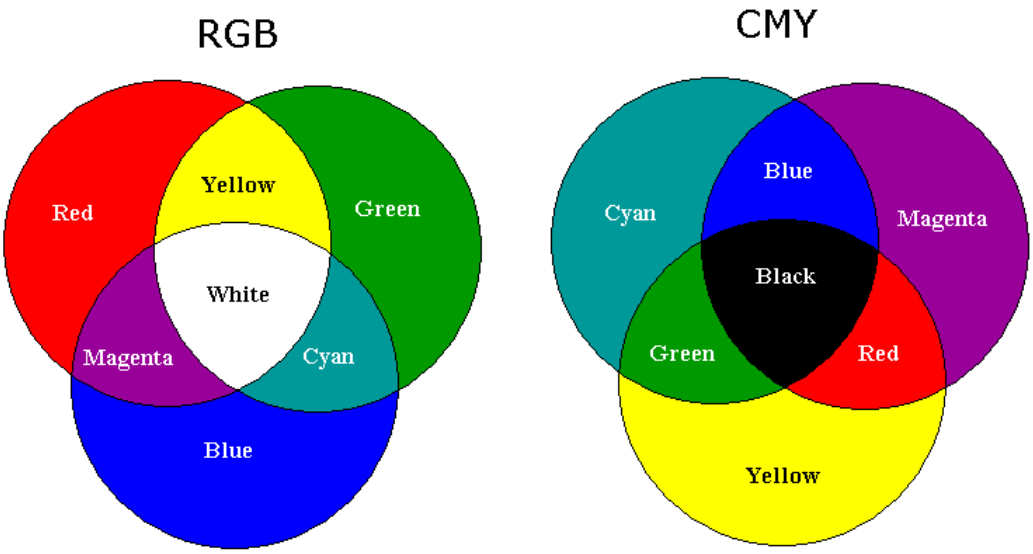


Рисунок 28 — Схема смешения цветов для моделей RGB и CMY.

Растровые дисплеи, как правило, используют аппаратно-ориентированную модель цветов RGB. Существуют также дисплеи с таблицей цветности, представляющей собой матрицу, каждый элемент которой — некоторый цвет (вектор RGB). В таких дисплеях значения кодов пикселей, заносимые в видеопамять, представляют собой индексы матрицы цветности. При отображении некоторого пикселя на экран по значению кода выбирается элемент таблицы цветности, содержащий тройку значений *R*, *G*, *B*. Эта тройка и передается на монитор для задания цвета пикселя на экране.

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

В моделях *RGB* и *CMY* легко задавать яркости для одного из основных цветов, но довольно затруднительно задать оттенок с требуемым цветовым тоном и насыщенностью, соответствующим какому-либо образцу цвета. В различного рода графических редакторах эта задача чаще всего решается с помощью интерактивного выбора из палитры цветов и формированием цветов в палитре путём подбора значений координат до получения требуемого визуального результата (рисунок 29). Иногда такая палитра наглядно отображает выбор вектора из цветового куба: сначала посредством одного движка выбирается цветовая плоскость, а затем на этой плоскости выбирается конкретная точка. Но и таким методом не сразу удается достигнуть желаемого эффекта, поскольку не так просто выбрать правильную цветовую плоскость.

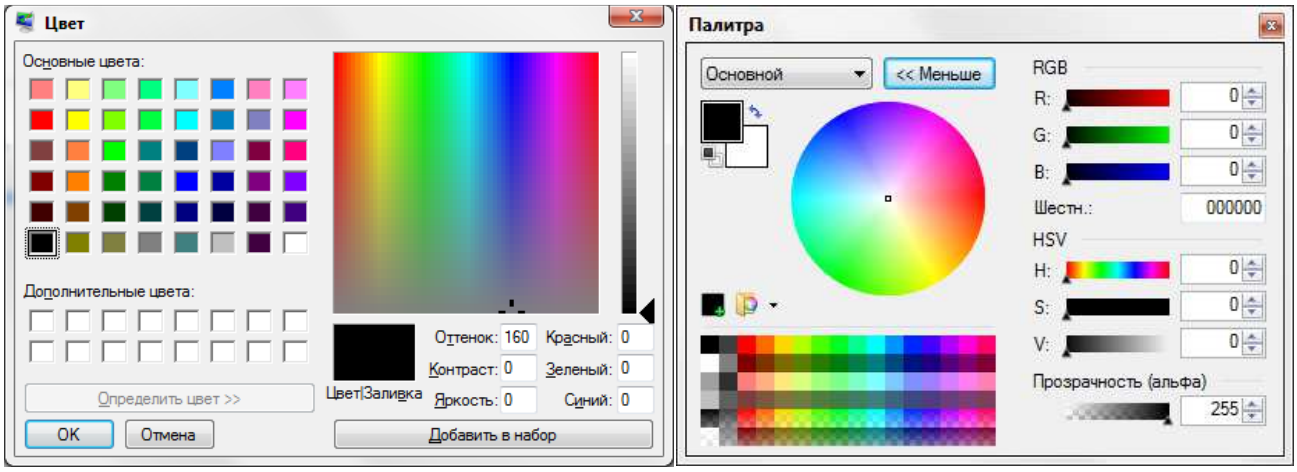


Рисунок 29 — Палитры цветов.

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

Поскольку для воспроизведения чёрного цвета требуется нанесение трёх красителей, а расходные материалы дороги, использование CMY-модели является не эффективным. Дополнительный фактор, не добавляющий привлекательности CMY-модели, — это появление нежелательных визуальных эффектов, возникающих за счёт того, что при выводе точки три базовые цвета могут ложиться с небольшими отклонениями. Поэтому к базовым трём цветам CMY-модели добавляют чёрный (blacK[20]) и получают новую цветовую модель *CMYK*.

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

$$\begin{aligned} K &= \min(C, M, Y); \\ C &= C - K; \\ M &= M - K; \\ Y &= Y - K. \end{aligned} \qquad (7)$$

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

---

[20] Одна из версий утверждает, что *K* — сокращение от англ. blacK. Чёрный не стали обозначать *B*, чтобы не путать с *B* из модели RGB. Согласно другому варианту, *K* является сокращением от слова "ключевой" — в англоязычных странах термином «key plate» обозначается печатная форма для чёрной краски печатаемая последней поверх заранее напечатанных трёх предыдущих красок. Печатники очень часто употребляют слово "Контур" относительно чёрного цвета. Так что возможно, что буква *K* в аббревиатуре — это сокращение от немецкого слова "Kontur". Если печатать только этой краской изображение получается скорее серое. Поэтому некоторые придерживаются мнения, что буква *K* в аббревиатуре означает "Kobalt" (темно-серый, нем.).

4 Цветовые модели HSV и HLS

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

В модели *HSV* (Hue, Saturation, Value — англ. — тон, насыщенность, количество света) используется цилиндрическая система координат, а множество допустимых цветов представляет собой шестигранный конус, поставленный на вершину (рисунок 30). По оси откладывается значение *V*, а расстояние от оси до боковой грани в горизонтальном сечении соответствует параметру *S* (за диапазон изменения этих величин принимается интервал от 0 до 1). Значение *S* равно 1, если точка лежит на боковой грани пирамиды.

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

Конус имеет единичную высоту ($V = 1$) и основание, расположенное в начале координат. В основании конуса величины *H* и *S* смысла не имеют. Белому цвету соответствует пара $S = 1$, $V = 1$. Ось *OV* ($S = 0$) соответствует ахроматическим цветам (серым тонам).

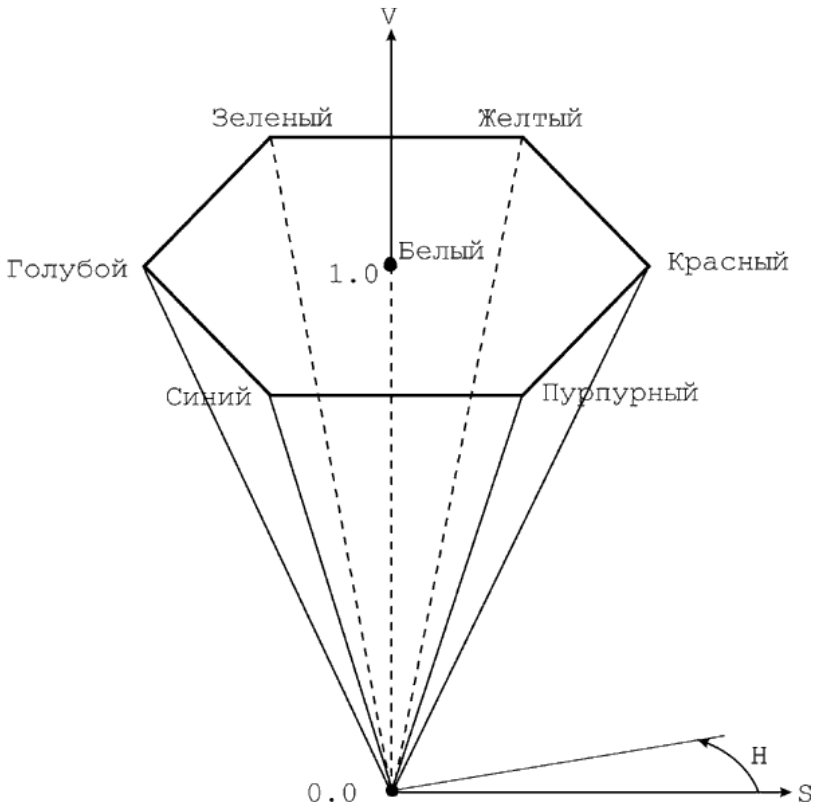


Рисунок 30 — Цветовое пространство HSV.

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

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

Ещё одним примером системы, построенной на интуитивных понятиях тона насыщенности и яркости, является система *HLS* (Hue, Lightness, Saturation — англ. — тон, светлота, насыщенность). Здесь множество всех цветов представляет собой два шестигранных конуса, поставленных друг на друга (рисунок 31).

Цветовая модель HLS является расширением модели HSV.

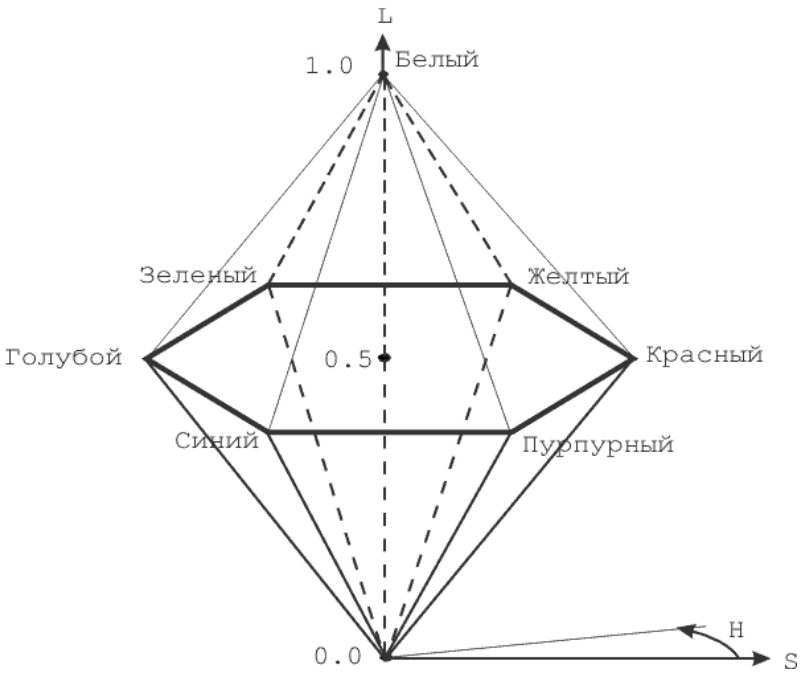


Рисунок 31 — Цветовое пространство HLS.

На рисунках 32 и 33 приведены блок-схемы преобразования моделей *HSV* и *HLS* в модель *RGB*.

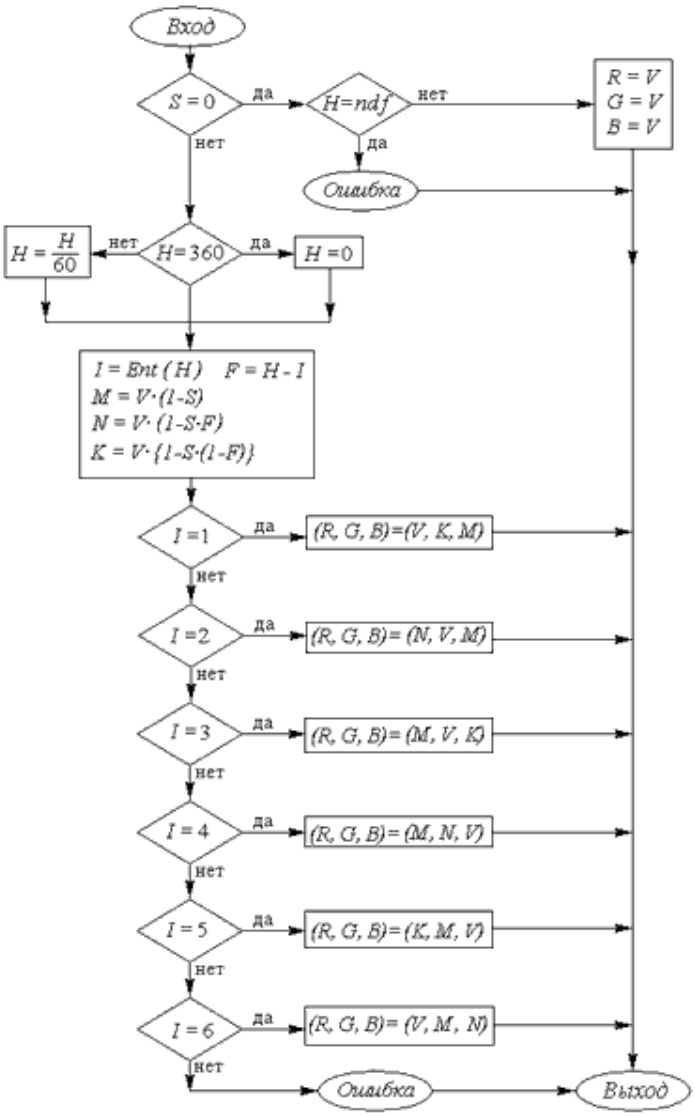


Рисунок 32 — Графическая схема алгоритма преобразования модели HSV в RGB.

В алгоритме преобразования модели HSV в RGB используется функция *Ent*, означающая целую часть числа. Кроме того, используется операция присваивания для векторов. Константа *ndf* (сокращенное от выражения "not defined") используется при входе в алгоритм для того, чтобы выяснить, задано ли значение переменной *H*. Например, по соглашению *ndf* может быть некоторым отрицательным значением, так как тон — это всегда положительная величина.

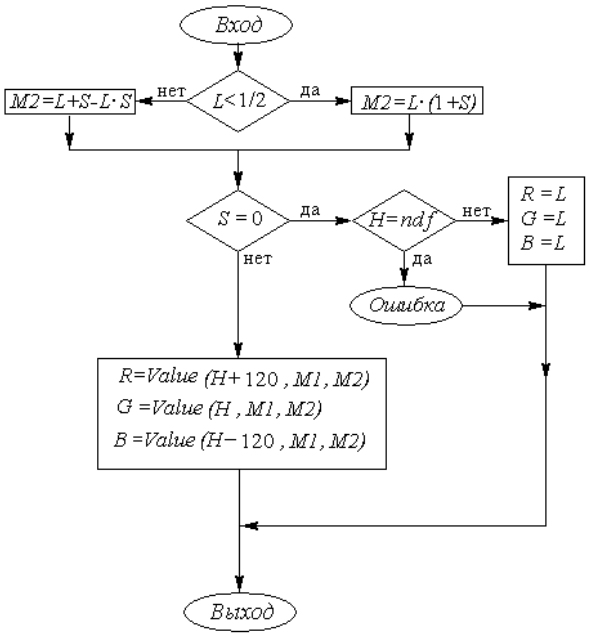


Рисунок 33 — Графическая схема алгоритма преобразования модели HLS в RGB.

В алгоритме преобразования модели HLS в RGB применяется вспомогательная функция *Value* (*H*, *M1*, *M2*) для вычисления значения компоненты *R*, *G* или *B* в зависимости от ситуации.

Алгоритм преобразования:

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

Раздел 2. Представление геометрической информации

Глава 3. Геометрические преобразования

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

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

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

1 Перенос

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

Параллельный перенос объекта сводится к перемещению всех его точек на одно и то же расстояние D в одном и том же направлении, заданном определённым вектором $\vec{v}$. Если вектор имеет длину D, то операция переноса может быть реализована путём сложения всех точек объекта с вектором $\vec{v}$. Понятно, что при такой операции сохраняются расстояния между точками и, как следствие, углы между отрезками, а также, что отрезки прямых перейдут в отрезки прямых (рисунок 34). Поэтому при переносе многоугольника нет необходимости подвергать этой операции бесконечное множество точек, достаточно просто перенести вершины, а затем соединить их отрезками.

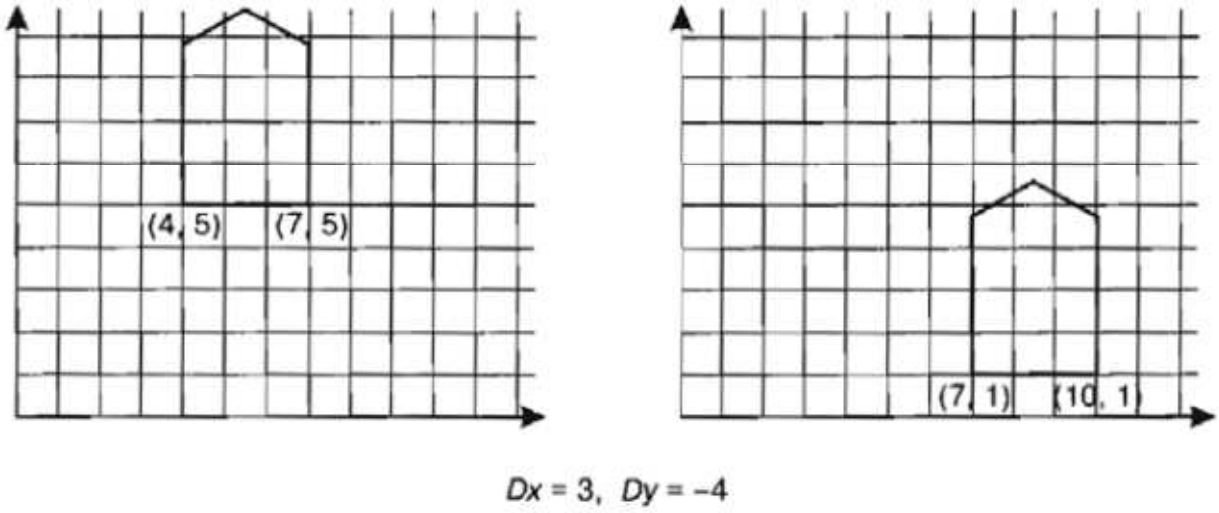


Рисунок 34 — Перенос объекта.

Математическую модель переноса опишем следующим образом.

Для каждой точки $P(x,y)$, которая перемещается в новую точку $P'(x',y')$, сдвигаясь на D_x единиц по оси X и на D_y по оси Y, можно написать:

$$x' = x + D_x,\ y' = y + D_y. \tag{8}$$

Определим векторы-строки:

$$P = (x\ y),\ P' = (x'\ y'),\ T = (D_x\ D_y). \tag{9}$$

Тогда уравнение можно записать в виде:

$$(x'\ y') = (x\ y) + (D_x\ D_y), \tag{10}$$

или кратко:

$$P' = P + T. \tag{11}$$

2 Масштабирование

Масштабирование объекта можно реализовать путём умножения координат всех его точек на некоторое число. Пусть имеются точки с координатами (x_1, y_1) и (x_2, y_2), над которыми выполняется такое преобразование. Результатом будут новые точки с координатами $(S_x\ x_1,\ S_y\ y_1)$ и $(S_x\ x_2,\ S_y\ y_2)$. Если $S_x = S_y = S_0$, понятно, что обе точки переместятся вдоль прямых, проходящих через саму точку и начало координат, т.е. в направлении своего же радиус-вектора (рисунок 35).

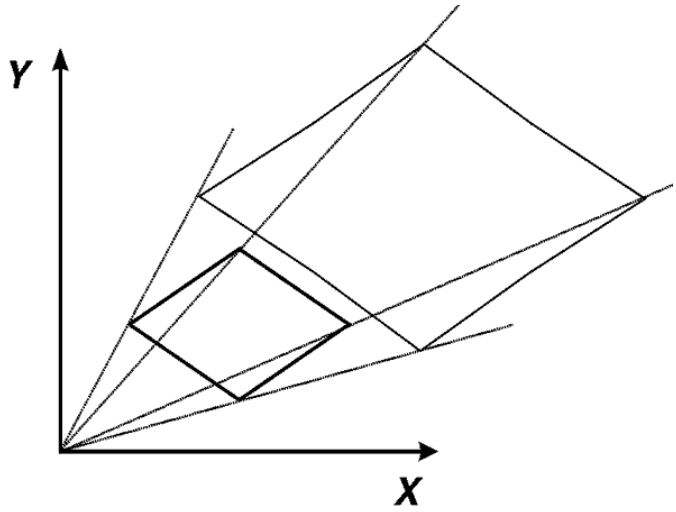


Рисунок 35 — Масштабирование с сохранением углов.

При этом расстояние между новыми точками будет в S_0 раз отличаться от прежнего, но углы между отрезками сохранятся. Ясно, что если коэффициент масштабирования $S_0 > 1$, соответствующий отрезок растягивается, а если меньше, то сжимается. Кроме того, при таком преобразовании объект смещается.

В случае, когда $S_x \neq S_y$, расстояния между точками изменятся неравномерно, поскольку растяжения в горизонтальном и вертикальном направлениях будут различными. Углы между отрезками также не сохранятся (рисунок 36).

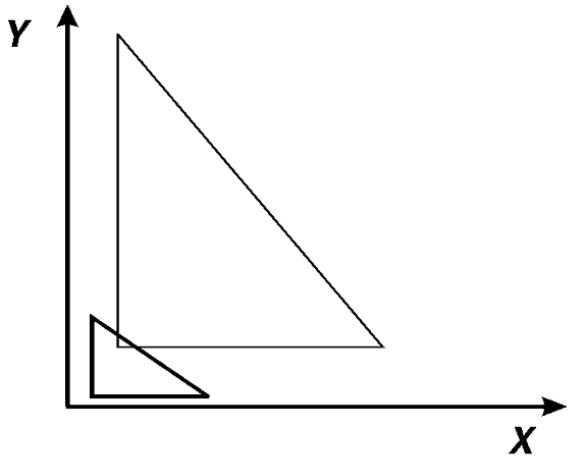


Рисунок 36 — Масштабирование с искажением углов.

Математическую модель данного преобразования опишем следующим образом.

Точки можно масштабировать в S_x раз по оси X и в S_y раз по оси Y, получив новые точки с помощью умножения:

$$x' = x\,S_x,\ y' = y\,S_y. \tag{12}$$

Определив S в виде матрицы:

$$S = \begin{pmatrix} S_x & 0 \\ 0 & S_y \end{pmatrix}, \tag{13}$$

можно записать в матричной форме:

$$(x'y') = (x\ y)\begin{pmatrix} S_x & 0 \\ 0 & S_y \end{pmatrix} \tag{14}$$

или

$$P' = P\,S. \tag{15}$$

3 Поворот

Вращения в плоскости перемещают точки по дуге окружности, центр которой находится в начале координат. Рассмотрим сначала движение одной точки при повороте на угол α (положительным является направление против часовой стрелки), т.е. поворот радиус-вектора на угол (рисунок 37). Пусть точка располагалась на расстоянии r от начала координат, а её радиус-вектор составлял угол β с осью абсцисс. Тогда координаты точки определяются формулами:

$$x = r\cos(\beta),\ y = r\sin(\beta) \tag{16}$$

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

$$\begin{aligned} x' &= r\cos(\beta+\alpha) = r\cos(\beta)\cos(\alpha) - r\sin(\beta)\sin(\alpha) = x\cos(\alpha) - y\sin(\alpha), \\ y' &= r\sin(\beta+\alpha) = r\sin(\beta)\cos(\alpha) + r\cos(\beta)\sin(\alpha) = x\sin(\alpha) + y\cos(\alpha). \end{aligned} \tag{17}$$

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

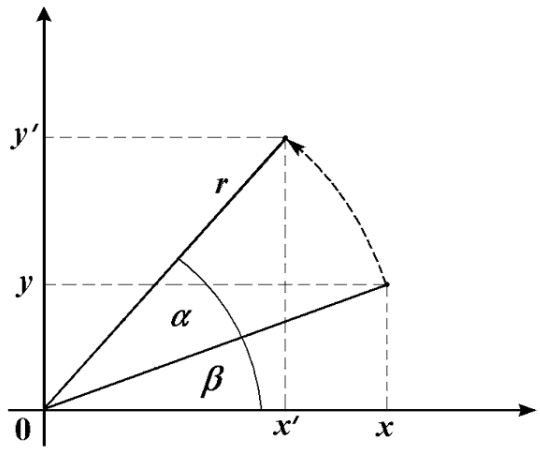


Рисунок 37 — Поворот на плоскости.

Так же, данное преобразование можно записать в матричной форме.

$$(x'y') = (x\ y)\begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix}, \tag{18}$$

или, определив матрицу поворота как:

$$R = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix}, \tag{19}$$

запишем:

$$P' = P\,R. \tag{20}$$

Глава 4. Представление геометрической информации

1 Геометрические примитивы

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

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

Во многих случаях для аппроксимации сложных поверхностей используются многогранники, но форма граней может быть различной. Пространственный многоугольник с числом вершин больше трёх не всегда бывает плоским, а в этом случае алгоритмы изображения многогранников могут привести к некорректному результату. Поэтому программист должен сам позаботиться о том, чтобы многогранник был описан правильно. В этом случае оптимальным выходом из положения является использование треугольников, поскольку треугольник всегда является плоским. В современной графике это, пожалуй, самый распространенный подход.

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

Любая графическая библиотека определяет свой набор примитивов. Так, например, широко распространенная интерактивная система трёхмерной графики OpenGL[21] включает в список своих примитивов точки (вершины), отрезки, ломаные,

---

[21] OpenGL (Open Graphics Library — открытая графическая библиотека, графический API[*]) — спецификация, определяющая независимый от языка программирования платформонезависимый программный интерфейс для написания приложений, использующих двумерную и трёхмерную компьютерную графику.

[*] Интерфейс программирования приложений (иногда интерфейс прикладного программирования) (англ. application programming interface, API [эй-пи-ай]) — набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) для использования во внешних программных продуктах. Используется программистами при написании всевозможных приложений.

многоугольники (среди которых особо выделяются треугольники и четырёхугольники), полосы (группы треугольников или четырёхугольников с общими вершинами) и шрифты. Кроме того, в неё входят и некоторые геометрические тела: сфера, цилиндр, конус и др.

Понятно, что для изображения таких примитивов должны быть разработаны эффективные и надежные алгоритмы, поскольку они являются конструктивными элементами. Исторически сложилось так, что первые дисплеи были векторными, поэтому базовым примитивом был отрезок. Однако, самая первая интерактивная программа Sketchpad А.Сазерленда [22] в качестве одного из примитивов имела прямоугольник, после чего этот объект уже традиционно входил в различные графические библиотеки.

1.1 Полигональные модели

Для этих пространственных моделей используются в качестве примитивов вершины (точки в пространстве), отрезки прямых (векторы), из которых строятся полилинии, полигоны и полигональные поверхности (рисунок 38). Главным элементом описания является вершина, все остальные являются производными. В трёхмерной декартовой системе координаты вершины определяются своими координатами (x, y, z), линия задается двумя вершинами, полилиния представляет собой незамкнутую ломаную линию, полигон — замкнутую ломаную линию. Полигон моделирует плоский объект и может описывать плоскую грань объёмного объекта. Несколько граней составляют этот объект в виде полигональной поверхности — многогранник или незамкнутую поверхность ("полигональная сетка").

В современной компьютерной графике векторно-полигональная модель является наиболее распространенной. Она применяется в системах автоматизированного проектирования, компьютерных играх, тренажёрах, геоинформационных системах (ГИС [23]), САПР и т. д. Достоинства этой модели заключаются в следующем:

---

[22] Айвен Эдвард Сазерленд (англ. Ivan Edward Sutherland, родился 16 мая 1938 в Хейстингсе, Небраска) — американский учёный в области информатики и пионер интернета. Получил премию Тьюринга от ACM в 1988 за создание «Sketchpad» — прообраза будущих САПР, имеющего ранний прототип графического интерфейса. Одновременно впервые применил объектно-ориентированный подход к программированию. В 2012 был удостоен премии Киото за ту же разработку.

[23] Геоинформационная система (географическая информационная система, ГИС) — система сбора, хранения, анализа и графической визуализации пространственных (географических) данных и связанной с ними информации о необходимых объектах.

- удобство масштабирования объектов;
- небольшой объём данных для описания простых поверхностей;
- аппаратная поддержка многих операций.

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

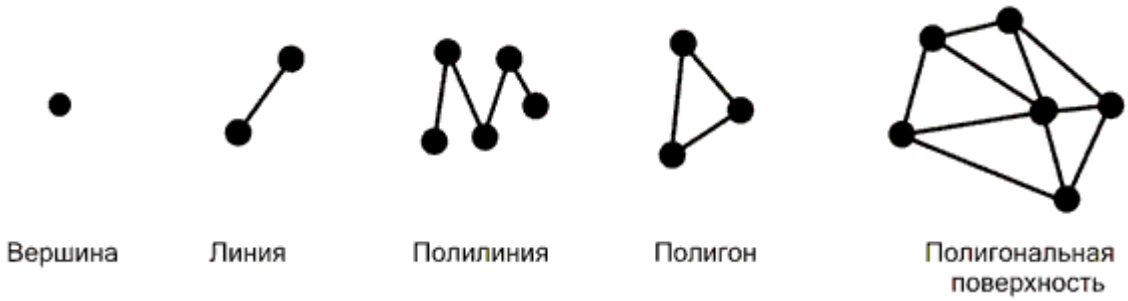


Рисунок 38 — Полигональная модель.

1.2 Воксельные модели

Воксельная модель — это представление объектов в виде трёхмерного массива объёмных (кубических) элементов (рисунок 39). Само название "воксель" составлено из двух слов: volume element (по аналогии с pixel — picture element). Так же как и пиксель, воксель имеет свои атрибуты (цвет, прозрачность и т.п.). Полная прозрачность вокселя означает пустоту в соответствующей точке объёма. Чем больше вокселей в определённом объёме и меньше их размер, тем точнее моделируются трёхмерные объекты.

Положительными чертами воксельной модели являются:

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

К её недостаткам относятся:

- большое количество информации, необходимое для представления объёмных данных;

- значительные затраты памяти, ограничивающие разрешающую способность, точность моделирования;
- проблемы при увеличении или уменьшении изображения; например, с увеличением ухудшается разрешающая способность изображения.

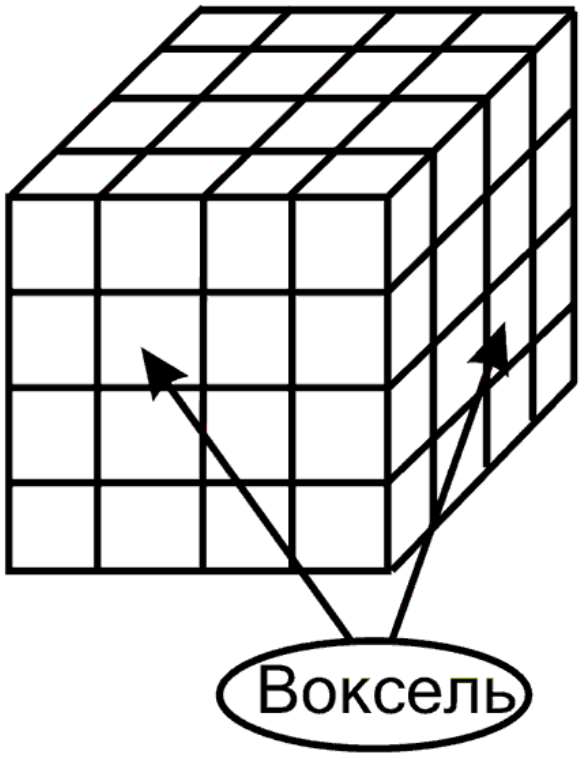


Рисунок 39 — Воксельная модель.

1.3 Поверхности свободных форм (функциональные модели)

Характерной особенностью данного способа задания поверхностей является то, что основным примитивом здесь является поверхность второго порядка — *квадрик*. Он определяется с помощью вещественной непрерывной функции трёх переменных (x, y, z) в виде неравенства $F(x, y, z) \geq 0$. Таким образом, квадрик — это замкнутое подмножество евклидова пространства, все точки которого удовлетворяют указанному неравенству. Уравнение $F(x, y, z) = 0$ описывает границу этого множества. Множество точек, удовлетворяющих неравенству $F(x, y, z) \leq 0$ образует внешнюю область квадрика.

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

- лёгкая процедура расчёта координат каждой точки;
- небольшой объём информации для описания достаточно сложных форм;
- возможность строить поверхности на основе скалярных данных без предварительной триангуляции[24].

2 Однородные координаты

2.1 Проективная геометрия

В первые десятилетия XIX века одновременно с успешным развертыванием исследований по основаниям геометрии возникла особая ветвь геометрических знаний — проективная геометрия. Источником её явились потребности графики и архитектуры.

Вначале проективная геометрия имела довольно ограниченный диапазон приложений. Но по мере роста она всё более и более проникала в различные геометрические области, а в конце XIX столетия исследования по проективной геометрии и по основаниям элементарной геометрии теснейшим образом объединились.

Замечательным результатом этого объединения было построение в рамках проективной геометрии глубокой теории, которая включила в единую схему геометрии Евклида, Лобачевского и Римана.

Известный французский геометр Понселе (1788 – 1867) выделил как объект исследования некоторые особые свойства геометрических фигур, названные им *проективными*.

Свойства проективных фигур.

Пусть Φ — произвольная фигура, расположенная в некоторой плоскости α; пусть β — какая-нибудь другая плоскость и пусть O — произвольная точка пространства, не содержащаяся ни в одной из двух плоскостей α и β (на рисунке 40 в качестве фигуры Φ выступает красный треугольник).

[24] Триангуляция — покрытие треугольниками.

Точка O вместе с каждой точкой M фигуры Φ определяет прямую OM. Прямая OM пересекает плоскость β в некоторой точке, которую обозначим через M' и будем называть проекцией точки M (из центра O на плоскость β).

Проекции всех точек фигуры Φ на плоскость β составят некоторую фигуру Φ', которая называется проекцией фигуры Φ (на рисунке 40 в качестве фигуры Φ' выступает синий треугольник).

Операция, с помощью которой получается фигура Φ', носит название центрального проектирования из точки O.

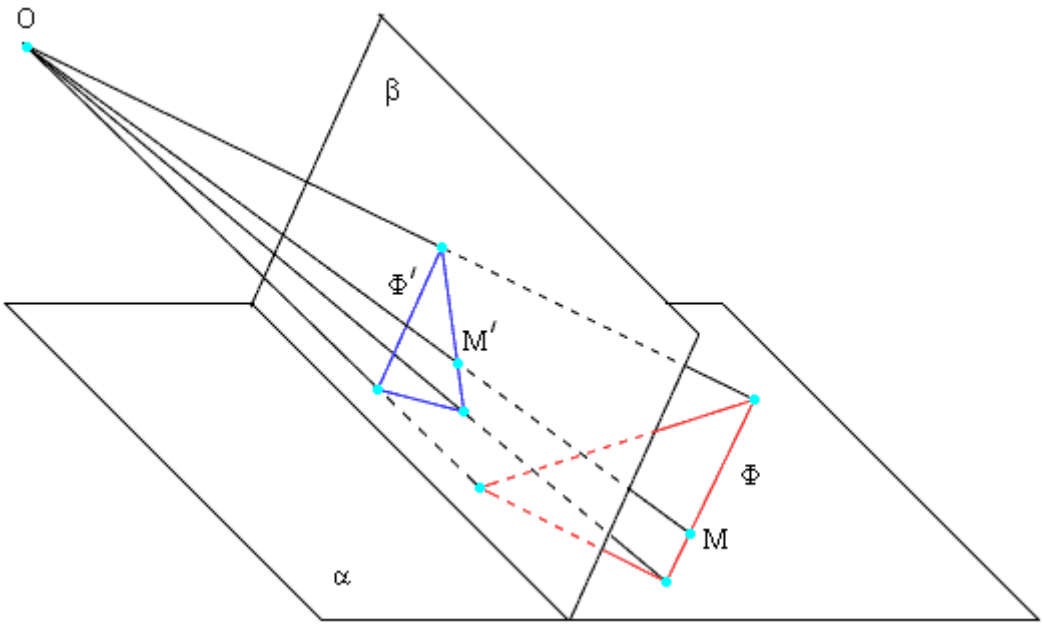


Рисунок 40 — Проекция фигуры.

Варьируя выбор точки O и плоскости β, можем получить при помощи центрального проектирования фигуры Φ бесконечное множество фигур, отчасти похожих на фигуру Φ, но во многих отношениях также и существенно от неё отличающихся.

Например, проектируя окружность, можно получить эллипс или параболу и даже гиперболу; проектируя правильный треугольник, можно получить треугольник произвольной формы, и т. д.

Таким образом, *многие свойства фигуры не переносятся на её проекцию*.

Так, свойства правильного треугольника могут не сохраниться при проектировании, в результате которого, вообще говоря, не будет получаться снова правильный треугольник; основное свойство окружности, выражаемое обычно её

определением, также может быть нарушено при проектировании, так как, проектируя окружность, можно получить, например, эллипс, и т. д.

Точно так же многие величины, связанные с фигурой, будут при проектировании, вообще говоря, меняться. Так, проектируя отрезок данной длины a, можно получить отрезок, длина которого как угодно велика или как угодно мала; проектируя треугольник данной площади Δ, можно получить треугольник, площадь которого будет больше или меньше величины Δ.

С другой стороны, фигуры обладают свойствами, которые сохраняются при любом проектировании, и с фигурами могут быть сопоставлены величины, также сохраняющиеся при любом проектировании. Такие свойства и величины называются *инвариантами проектирования*.

Именно эти свойства фигур, инвариантные по отношению к любому проектированию, Понселе назвал *проективными свойствами*, рассматривая их как объекты исследования в проективной геометрии. Кроме того, объектами проективной геометрии являются инвариантные относительно проектирований величины.[25]

Задача изучения проективных свойств фигур привлекла к себе внимание многих геометров, среди которых после Понселе можно отметить Шаля (1793 – 1880) и Штейнера (1769 – 1863). Им принадлежит разработка ряда общих вопросов проективной геометрии, в которых Штейнер, Шаль и сопутствующие им геометры видели возрождение синтетического направления в геометрии.

Развивая синтетические методы в противовес аналитическим, эти геометры достигли известных успехов в усовершенствовании аппарата проективной геометрии и в применении его к различным геометрическим задачам.

---

[25] Примеры:

Если точки $P_1, P_2, \ldots, P_n$ фигуры Φ лежат на прямой, то проекции этих точек $P'_1, P'_2, \ldots, P'_n$ также лежат на некоторой прямой. Следовательно, свойство точек фигуры, заключающееся в прямолинейном их расположении, является проективным. Иначе ещё можно сказать, что прямая есть объект проективной геометрии (см. рисунок 40).

Если точки $Q_1, Q_2, \ldots, Q_n$ фигуры Φ лежат на каком-нибудь коническом сечении k, то проекции этих точек $Q'_1, Q'_2, \ldots$, Q'n также лежат на некотором коническом сечении k'. Иначе говоря, коническое сечение есть объект проективной геометрии (см. рисунок 40).

При этом только нужно иметь в виду, что свойства, присущие исключительно окружности, или исключительно эллипсу, или исключительно параболе, или исключительно гиперболе, не являются проективными свойствами, поэтому в проективной геометрии не делается различия между коническими сечениями, как в элементарной геометрии.

Иными словами, хотя конические сечения суть объекты проективной геометрии, но отдельные виды их — окружности, эллипсы, параболы, гиперболы — в проективной геометрии не различаются и порознь не исследуются.

Однако принципиальное значение проективной геометрии в развитии геометрических идей определяется отнюдь не количеством отдельных случаев, где её методы оказываются удобнее методов аналитической геометрии.

Сейчас видно это значение в той общности проективной геометрии, которая позволяет соединить воедино разные геометрические системы, в частности, включить в проективную схему и элементарную геометрию.

Между тем у Штейнера и Шаля проективная геометрия выглядела как часть элементарной. Превращение проективной геометрии во вполне самостоятельную дисциплину явилось уже делом второй половины XIX столетия.

Важной предпосылкой для этого превращения было употребление в проективной геометрии бесконечно удалённых геометрических элементов.

Бесконечно удалённые элементы.

Пусть A — произвольная точка пространства и a — прямая, не проходящая через точку A (см. рисунок 41). Проведём через A и a плоскость α и будем рассматривать всевозможные прямые плоскости α, проходящие через точку A. Эти прямые составляют плоский пучок с центром A; называемый *пучок A*.

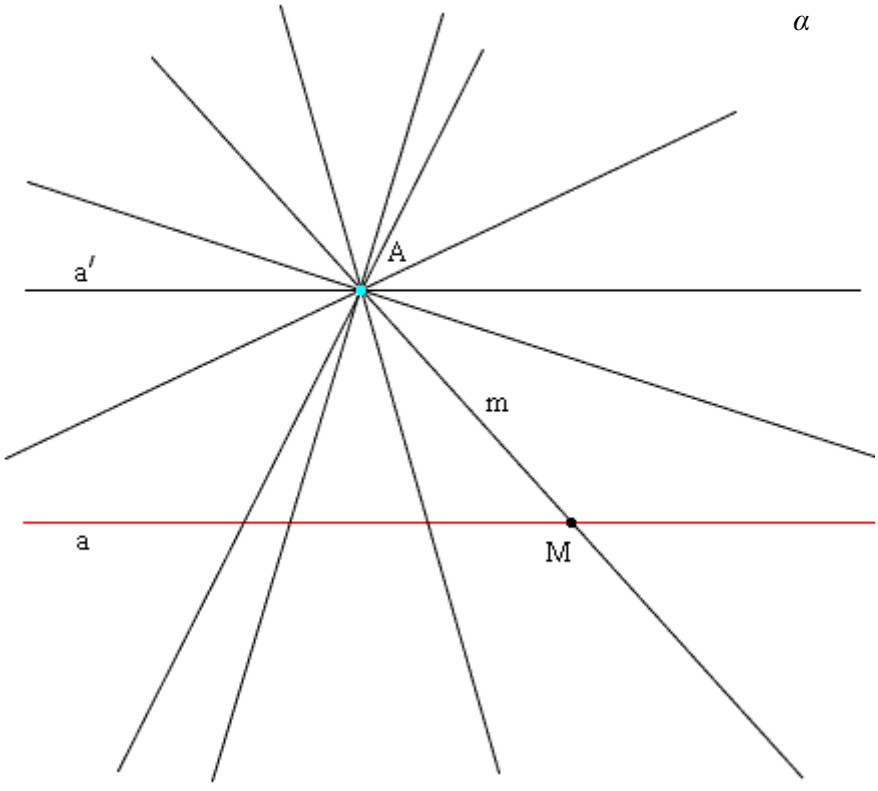


Рисунок 41 — Проекция точки.

Между лучами этого пучка и точками прямой a установим соответствие, сопоставляя с каждой точкой M прямой a тот луч m пучка A, который пересекает прямую a в точке M. Луч m называют *лучом, проектирующим точку M.*

Очевидно, что как бы ни была расположена точка M на прямой a, ей всегда соответствует определённый луч m. Но нельзя утверждать, что любому лучу пучка A соответствует точка прямой a.

Именно, луч a' пучка A, параллельный прямой a, не пересекает её и поэтому не имеет соответствующей себе точки.

Таким образом, соответствие между лучами пучка A и точками прямой a *не является взаимно однозначным*.

Это обстоятельство при исследовании проектирований служит источником многочисленных неудобств. Чтобы избежать их, условливаются рассматривать параллельные прямые как пересекающиеся на бесконечности.

Тогда луч a' в пучке A, параллельный прямой a, как и всякий другой луч пучка, будет иметь на прямой a соответствующую себе точку, но только не обыкновенную точку, а некоторый новый объект, называемый *бесконечно удалённой точкой* прямой a.

Бесконечно удалённая точка прямой считается принадлежащей также каждой плоскости, которая проходит через эту прямую.

Далее полагают, что параллельные прямые имеют одну общую бесконечно удалённую точку; соответственно этому систему параллельных прямых, расположенных в одной плоскости, называют пучком с бесконечно удалённым центром.

Бесконечно удалённые точки непараллельных прямых считаются различными. Таким образом, каждая плоскость содержит бесконечно много различных бесконечно удалённых точек.

Совокупность всех бесконечно удалённых точек плоскости называют её бесконечно удалённой прямой.

Совокупность всех бесконечно удалённых точек пространства называют бесконечно удалённой плоскостью.

Такая терминология оправдывается следующими двумя обстоятельствами:

- две параллельные плоскости имеют общие бесконечно удалённые точки, вследствие чего совокупность бесконечно удалённых точек плоскости можно рассматривать как образ, получаемый при пересечении двух плоскостей; поэтому совокупность бесконечно удалённых точек плоскости естественно назвать прямой;
- множество всех бесконечно удалённых точек пространства при пересечении с любой обыкновенной плоскостью определяет бесконечно удалённую прямую. Поэтому указанное множество естественно назвать плоскостью.

Все изложенное можно резюмировать следующим образом.

Множество объектов евклидова пространства дополняется новыми элементами, которым дается название "бесконечно удалённая точка", "бесконечно удалённая прямая", "бесконечно удалённая плоскость".

Присоединение новых элементов производится с соблюдением определённых условий, именно:

- ко множеству точек каждой прямой прибавляется одна бесконечно удалённая точка; к множеству прямых каждой плоскости прибавляется одна бесконечно удалённая прямая; к множеству плоскостей пространства прибавляется одна бесконечно удалённая плоскость;
- свойства взаимной принадлежности расширенного множества геометрических элементов должны удовлетворять требованиям всех аксиом сочетания (т.е. первой группы аксиом Гильберта);
- свойства взаимной принадлежности расширенного множества геометрических элементов должны быть таковы, что каждые две плоскости имеют общую прямую, каждые прямая и плоскость имеют общую точку, а также имеют общую точку каждые две прямые, расположенные в одной плоскости.

Прямая, дополненная бесконечно удалённой точкой, называется *проективной прямой*. Проективную прямую следует представлять себе в виде замкнутой линии. Плоскость, дополненная бесконечно удалённой прямой, называется проективной

плоскостью. Пространство, дополненное бесконечно удалённой плоскостью, называется *проективным пространством*.[26]

Бесконечно удалённые элементы нередко вводятся в рассмотрение и в элементарной геометрии. Но в элементарной геометрии использование их по существу ограничивается лишь особой манерой словесного выражения геометрических фактов (вместо того, чтобы говорить, что прямые параллельны, их называют сходящимися в бесконечности, цилиндр рассматривают как конус с бесконечно удалённой вершиной и т.д.).

В проективной геометрии, напротив, бесконечно удалённые элементы играют такую же роль, как и обыкновенные геометрические образы, и являются органичной частью проективного пространства.

Причина такого различия станет вполне ясной, если сравнить объекты исследования элементарной геометрии и проективной геометрии. Элементарная геометрия в значительной степени посвящена изучению так называемых метрических свойств фигур, т.е. таких свойств, которые связаны с измерением геометрических величин (длин, углов и площадей).

Измерение любого отрезка AB с обыкновенными концами всегда возможно и приводит в результате к определённому числу, выражающему длину отрезка AB.

Но в том случае, когда один конец отрезка является бесконечно удалённой точкой, процесс измерения теряет смысл, так как на таком отрезке линейная единица откладывается бесконечно много раз.

Точно так же процесс измерения углов неприменим в том случае, когда одна сторона угла есть бесконечно удалённая прямая, и "наивные" способы измерения площадей неприменимы к фигурам, содержащим бесконечно удалённые элементы.

Таким образом, в элементарной геометрии бесконечно удалённые элементы по необходимости играют особую роль и по свойству своих отношений к обыкновенным геометрическим элементам существенно от них отличаются.

В противоположность этому, в проективной геометрии, поскольку метрические свойства фигур не являются её объектами, указанные выше

---

[26] См. лист Мёбиуса.

обстоятельства, отличающие бесконечно удалённые элементы от остальных, теряют силу.

Более того, так как при проектированиях бесконечно удалённые элементы могут переходить в обыкновенные, то, следовательно, они не обладают никакими проективными свойствами, которые отличали бы их от обыкновенных элементов. Поэтому в проективной геометрии различия между обыкновенными и бесконечно удалёнными элементами нет.

Идея бесконечно удалённых элементов возникла довольно давно. Но равноправие бесконечно удалённых и обыкновенных элементов, естественное с точки зрения проективной геометрии, оставалось иллюзорным, пока проективные свойства фигур исследовались методами элементарной геометрии, так как эти методы опираются на измерение, а метрика элементарной геометрии обязательно приводит к различию между конечными и бесконечными удалёнными образами.

Чтобы придать понятию проективного пространства точный смысл, оказалось необходимым полностью изгнать из проективной геометрии всё, что связано с измерением.

Задача освобождения проективной геометрии от использования измерений в принципе была решена Штаудтом (1798 – 1867).

Проективная геометрия, освобожденная от метрики, превратилась в дисциплину, изучающую только свойства взаимного расположения геометрических образов.

Вместе с тем проективная геометрия сделалась самостоятельной геометрической дисциплиной со своей собственной аксиоматикой и собственной совокупностью объектов (каковыми являются проективная прямая, проективная плоскость и проективное пространство).

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

В случае последовательного выполнения любой комбинации операций вращения и масштабирования результат легко можно записать в виде произведения матриц соответствующих преобразований. Это будет матрица результирующего поворота и масштабирования. Очевидно, что удобнее применять результирующую матрицу вместо того, чтобы каждый раз заново вычислять произведение матриц. Однако, таким способом нельзя получить результирующую матрицу

преобразования, если среди последовательности преобразований присутствует хотя бы один перенос. Матричное произведение в компьютерной графике также называют *композицией*. Было бы удобнее иметь математический аппарат, позволяющий включать в композиции преобразований все три операции (масштабирование, поворот, перенос). При этом получился бы значительный выигрыш в скорости вычислений. Однородные координаты и есть этот математический аппарат.

Однородные координаты — мощный математический инструмент, находящий своё применения в различных разделах компьютерной графики — геометрическом моделировании, визуализации, машинном зрении и т.д. Однородные координаты явно или неявно используются в любом графическом пакете на этапах преобразования и затенения геометрии. Например, в OpenGL или DirectX[27].

Однородные координаты — координаты, обладающие тем свойством, что определяемый ими объект не меняется при умножении всех координат на одно и то же ненулевое число.

Привычный аппарат декартовых координат, не подходит для решения некоторых важных задач в силу следующих соображений:

- в декартовых координатах невозможно описать бесконечно удалённую точку. А многие математические и геометрические концепции значительно упрощаются, если в них используется понятие бесконечности. Например, "бесконечно удалённый источник света";
- с точки зрения алгебраических операций, декартовы координаты не позволяют провести различия межу точками и векторами в пространстве. Действительно, (1, -2, 5) — это направление или точка(?);
- невозможно использовать унифицированный механизм работы с матрицами для выражения преобразований точек. С помощью матриц [3 x 3] можно описать вращение и масштабирование, однако описать смещение ($x = x + a$) нельзя;
- аналогично, декартовы координаты не позволяют использовать матричную запись для задания перспективного преобразования (проекции) точек.

[27] DirectX (от англ. direct — прямо, напрямую) — это набор API, разработанных для решения задач, связанных с программированием под Microsoft Windows. Наиболее широко используется при написании компьютерных игр.

Как рассматривалось ранее, двумерные преобразования имеют различный вид. Это различие затрудняет формирование суммарного преобразования и устраняется использованием двумерных однородных координат точки, имеющих вид: (x, y, w). Здесь w — произвольный множитель, не равный 0. Число w так же называется *масштабным множителем*.

Двумерные декартовые координаты точки получаются из однородных делением на множитель w:

$$X = x / w;\ Y = y / w. \tag{21}$$

В иной форме записи преобразование из однородных координат в обычные:

$$(x/w, y/w, w/w) \equiv (X, Y, 1) \tag{22}$$

Однородные координаты для двумерного случая можно представить как промасштабированные с коэффициентом w значения двумерных координат, расположенные в плоскости с $Z = w$.

В общем случае осуществляется переход от n-мерного пространства к $(n+1)$-мерному. Обратное преобразование называется проекцией однородных координат.

Некоторые свойства однородных координат:

- два набора чисел, представляющих однородные координаты, соответствуют одной точке декартового пространства, если они могут быть получены один из другого умножением на некоторый множитель. ((2, 5, 3) и (4, 10, 6) представляют одну точку);
- в силу произвольности значения w в однородных координатах не существует единственного представления точки, заданной в декартовых координатах;
- как минимум одно число из тройки должно быть отлично от нуля — точка (0, 0, 0) не будет определена;
- если $w \neq 0$, то деление на неё даст декартовы координаты $(x/w, y/w, 1)$;
- если $w = 0$, то точка находится в бесконечности.

Для пояснения:

Рассмотрим случай двумерного пространства. Каждая точка плоскости с координатами (x, y) может одновременно рассматриваться как точка трёхмерного пространства с координатами $(x, y, 1)$, т.е. как точка, лежащая на плоскости $z = 1$. С

другой стороны, каждой точке трёхмерного пространства (x, y, z) при условии $z \neq 0$ соответствует единственная точка этой же плоскости $\left(\frac{x}{z}, \frac{y}{z}, 1\right)$. При этом получается, что каждой точке плоскости $(x, y, 1)$ соответствует прямая, проходящая через начало координат, т.е. устанавливается взаимно однозначное соответствие между точками плоскости и множествами (tx, ty, t), $-\infty < t < \infty$, $t \neq 0$ (рисунок 42).

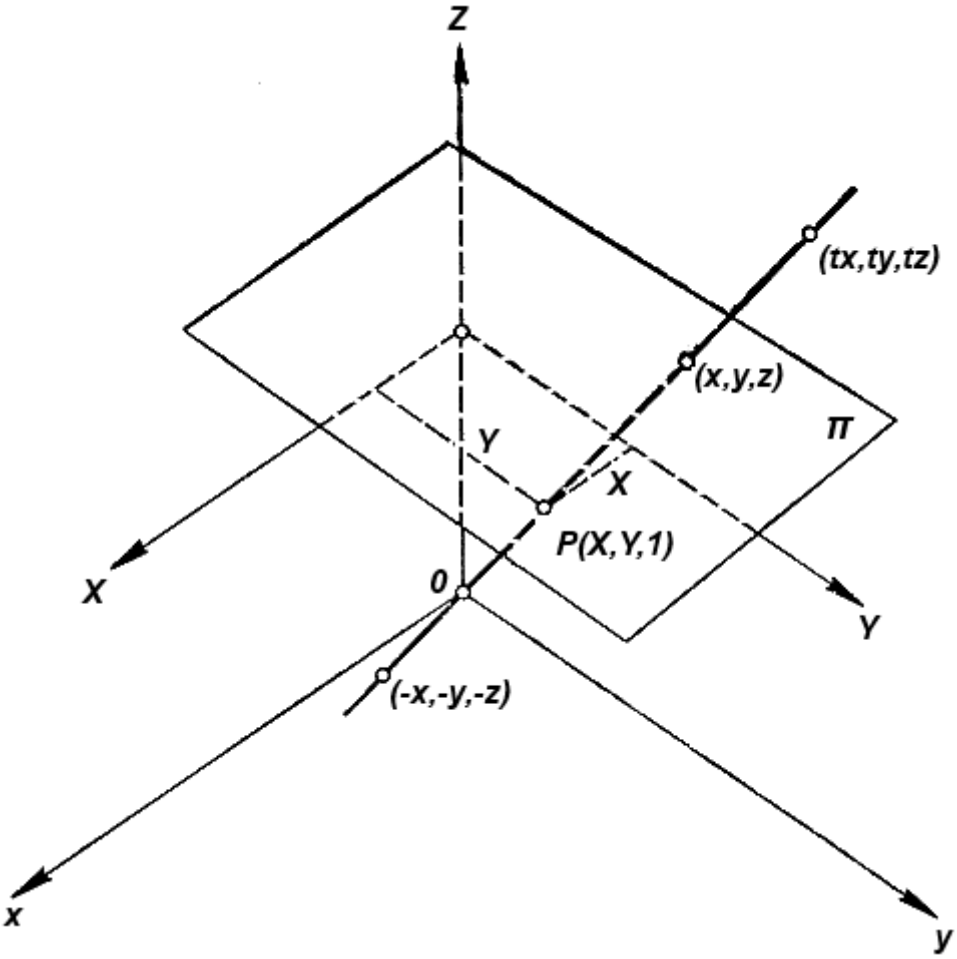


Рисунок 42 — Однородные координаты.

Если теперь рассматривать точку плоскости как принадлежащую трёхмерному пространству, то её двумерные преобразования можно будет описывать с помощью матриц [3 x 3], причём можно будет задавать таким способом не только повороты и масштабирование, но и сдвиги и проекции (как ортографические, так и центральные).

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

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}. \quad (23)$$

Операция масштабирования может быть записана в виде:

$$\begin{pmatrix} S_x x \\ S_y y \\ 1 \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}. \quad (24)$$

Перенос на вектор (D_x, D_y) также можно задать с помощью матрицы:

$$\begin{pmatrix} x + D_x \\ y + D_y \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & D_x \\ 0 & 1 & D_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}. \quad (25)$$

Проекции точки на оси координат определяются с помощью *матриц проекции*:

$$P_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, P_y = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \quad (26)$$

В трёхмерном пространстве каждой точке (x, y, z) будет ставиться точка четырёхмерного пространства $(x, y, z, 1)$, соответственно для основных преобразований используется матрица [4 x 4]. Матрицы строятся аналогично, как это делалось в двумерном случае.

Матрица сдвига на вектор (D_x, D_y, D_z) имеет вид:

$$P = \begin{pmatrix} 1 & 0 & 0 & D_x \\ 0 & 1 & 0 & D_y \\ 0 & 0 & 1 & D_z \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad (27)$$

Матрица масштабирования:

$$S = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad (28)$$

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

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

$$R_z = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad (29)$$

Проекции точек на координатные плоскости осуществляются с помощью матриц:

$$P_{xy} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, P_{yz} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, P_{zx} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad (30)$$

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

Каждое из этих преобразований реализуется с помощью матрицы, а полное преобразование можно выполнить, используя произведение матриц[28].

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

---

[28] Пусть даны две прямоугольные матрицы *A* и *B* размерности *m* x *n* и *n* x *q* соответственно:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, B = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1q} \\ b_{21} & b_{22} & \cdots & b_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nq} \end{pmatrix}.$$

Тогда матрица *C* размерностью *m* x *q* называется их произведением:

$$C = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1q} \\ c_{21} & c_{22} & \cdots & c_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mq} \end{pmatrix},$$

где:

$$c_{ij} = \sum_{r=1}^{n} a_{ir} b_{rj} \ (i = 1, 2, \ldots, m;\ j = 1, 2, \ldots, q).$$

Операция умножения двух матриц выполнима только в том случае, если число столбцов в первом сомножителе равно числу строк во втором; в этом случае говорят, что форма матриц *согласована*. В частности, умножение всегда выполнимо, если оба сомножителя — квадратные матрицы одного и того же порядка.

Следует заметить, что из существования произведения *A·B* вовсе не следует существование произведения *B·A*.

$$x' = x\cos\alpha - y\sin\alpha,$$
$$y' = x\sin\alpha + y\cos\alpha, \qquad (31)$$

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

Пример графического преобразования с помощью матриц

Приведенный ниже пример показывает геометрические преобразования фигуры (квадрата) с использованием матриц преобразования, с переходом из декартовых координат в однородные и обратно. Все расчёты сделаны в математическом пакете, показывающий логику и алгоритм действий для построения преобразований. Следует учесть, что не все действия, выполняемые в математическом пакете, также однозначно реализуются в языках программирования (например, перемножение матриц).

Пример геометрического преобразования 2-D изображения.

Зададим матрицу M = [5 x 2] координат фигуры преобразования (рисунок П.1), условимся, что 1-ый столбец (0-вой по счёту) — координаты по оси X, 2-ой (1-ый по счёту) — координаты по оси Y:

$$M := \begin{pmatrix} 1 & 1 \\ 3 & 1 \\ 3 & 3 \\ 1 & 3 \\ 1 & 1 \end{pmatrix}$$

Зададим матрицы преобразований, где D — матрица перемещений, S — масштабирования, R — вращения:

$D_x := 5 \qquad D_y := 5 \qquad S_x := 2 \qquad S_y := 2 \qquad \alpha := \frac{\pi}{4}$

$$D := \begin{pmatrix} 1 & 0 & D_x \\ 0 & 1 & D_y \\ 0 & 0 & 1 \end{pmatrix} \qquad S := \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad R := \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

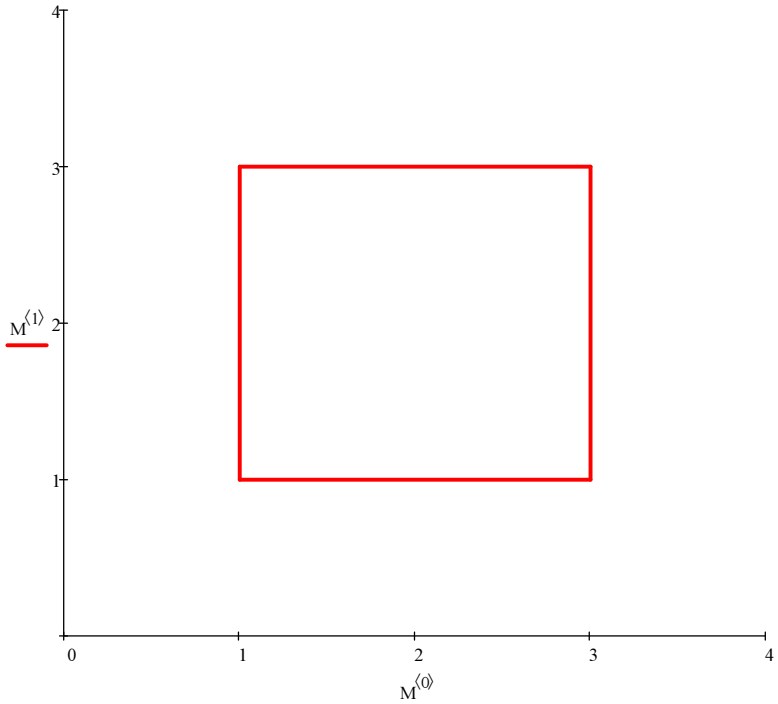


Рисунок П.1 — Исходное изображение (квадрат).

Преобразуем координаты фигуры в однородные:

первая точка: вторая точка: и т.д.

$$M'_0 := \begin{pmatrix} M_{0,0} \\ M_{0,1} \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \qquad M'_1 := \begin{pmatrix} M_{1,0} \\ M_{1,1} \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix}$$

Для того чтобы *не* преобразовывать каждую точку фигуры вручную, создадим программный фрагмент:

```
M' := | for i ∈ 0..4
      |   for j ∈ 0..1
      |     m''j,0 ← Mi,j
      |   m''2,0 ← 1
      |   m'i ← m''
      | M' ← m'
```

В итоге получаем вектор векторов *M′* (одномерный массив, каждый элемент которого является одномерным массивом [3 x 1]) с однородными координатами каждой точки:

$$M' = \begin{pmatrix} \{3,1\} \\ \{3,1\} \\ \{3,1\} \\ \{3,1\} \\ \{3,1\} \end{pmatrix}$$

Проверим себя, выведя координаты 2-ой точки фигуры (1-ая по счёту):

$$M'_1 = \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix}$$

Далее создадим программный фрагмент для преобразования фигуры в однородных координатах, который заключается в перемножении матрицы преобразования (параметр *Tr*) на исходную матрицу с однородными координатами (*M′*):

```
M'T(Tr) := | for i ∈ 0..4
           |   m'' ← Tr·M'i
           |   m'i ← m''
           | M'T ← m'
```

Результатом будет являться вектор M'_T, каждый элемент которого содержит вектор с преобразованными однородными координатами:

$$M'_T(Tr) \rightarrow \begin{pmatrix} \{3,1\} \\ \{3,1\} \\ \{3,1\} \\ \{3,1\} \\ \{3,1\} \end{pmatrix}$$

Теперь необходимо преобразовать матрицу однородных координат в матрицу декартовых (избавиться от 1), т.е.:

первая точка: вторая точка: и т.д.

$$M_{T_{0,0}} := \left(M'_T(Tr)_0\right)_0 \qquad M_{T_{0,1}} := \left(M'_T(Tr)_0\right)_1$$

$$M_{T_{1,0}} := \left(M'_T(Tr)_1\right)_0 \qquad M_{T_{1,1}} := \left(M'_T(Tr)_1\right)_1$$

Для того чтобы *не* преобразовывать каждую точку фигуры вручную, создадим программный фрагмент:

$$M_T(Tr) := \left| \begin{array}{l} \text{for } i \in 0..4 \\ \quad \left| \begin{array}{l} \text{for } j \in 0..1 \\ \quad m'_{i,j} \leftarrow \left(M'_T(Tr)_i\right)_j \end{array} \right. \\ \quad m \leftarrow m' \\ M_T \leftarrow m \end{array} \right.$$

Результатом такого программного фрагмента будет являться матрица координат M_T, где 1-ый столбец (0-вой по счёту) будет содержать координаты по оси X, а 2-ой (1-ый по счёту) — координаты по оси Y.

Правильность преобразований можно проверить, задав в качестве параметра 1 (единицу), таким образом, получив исходную матрицу:

$$M_T(1) = \begin{pmatrix} 1 & 1 \\ 3 & 1 \\ 3 & 3 \\ 1 & 3 \\ 1 & 1 \end{pmatrix} \qquad M = \begin{pmatrix} 1 & 1 \\ 3 & 1 \\ 3 & 3 \\ 1 & 3 \\ 1 & 1 \end{pmatrix}$$

Матрицы с координатами при перемещении (*D*), масштабировании (*S*) и вращении (*R*) вычисляются, подставив в качестве параметра *Tr* необходимую матрицу:

$$M_T(D) = \begin{pmatrix} 6 & 6 \\ 8 & 6 \\ 8 & 8 \\ 6 & 8 \\ 6 & 6 \end{pmatrix} \quad M_T(S) = \begin{pmatrix} 2 & 2 \\ 6 & 2 \\ 6 & 6 \\ 2 & 6 \\ 2 & 2 \end{pmatrix} \quad M_T(R) = \begin{pmatrix} 0 & 1.414 \\ 1.414 & 2.828 \\ 0 & 4.243 \\ -1.414 & 2.828 \\ 0 & 1.414 \end{pmatrix}$$

Рассчитав все координаты, изобразим полученные преобразования на плоскости (рисунок П.2).

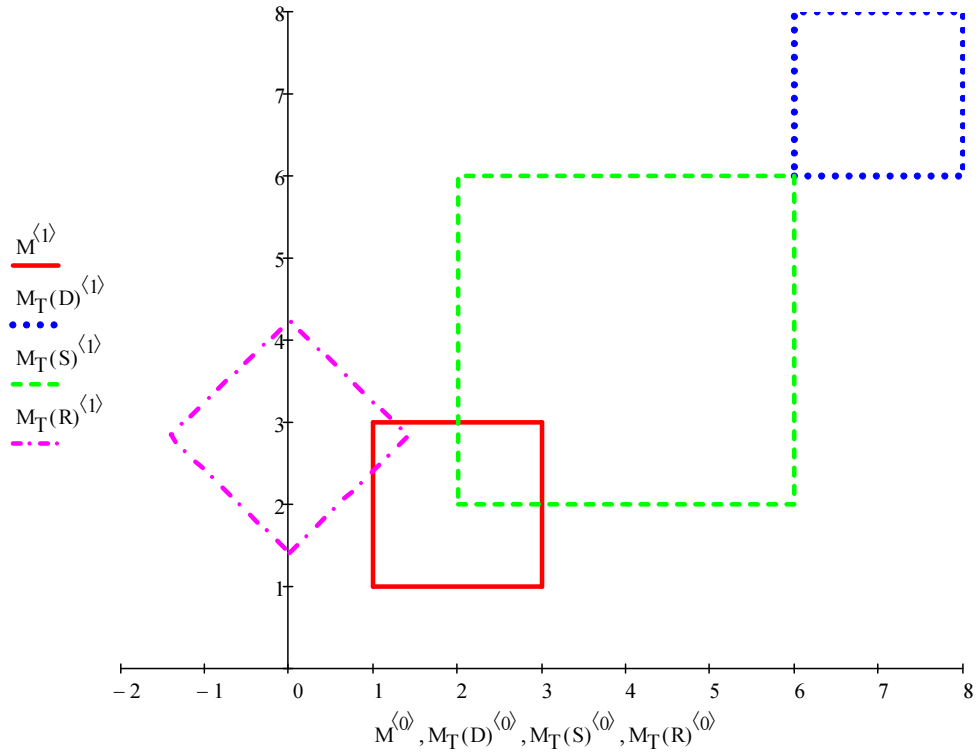


Рисунок П.2 — Геометрические преобразования фигуры.

Задача:

Дана матрица исходной фигуры M и результирующая матрица после преобразования масштабированием — M'. Найти матрицу масштабирования S.

$$M = \begin{pmatrix} 0 & 0 \\ 0 & 2 \\ 2 & 2 \\ 0 & 0 \end{pmatrix}, M' = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 4 & 0 \\ 0 & 0 \end{pmatrix};$$

$$S = (?).$$

Пример геометрического преобразования 3-D изображения.

Зададим матрицу M = [16 x 3] координат фигуры преобразования (рисунок П.3), условимся, что 1-ый столбец (0-вой по счёту) — координаты по оси X, 2-ой (1-ый по счёту) — координаты по оси Y, 3-ий — по оси Z:

M =

	0	1	2
0	1	1	2
1	3	1	2
2	3	3	2
3	1	3	2
4	1	1	2
5	1	1	3
6	3	1	3
7	3	1	2
8	3	3	3
9	3	3	2
10	1	3	3
11	1	3	2
12	1	1	3
13	1	3	3
14	3	3	3
15	3	1	3

Пусть r — количество строк матрицы M (равное 16), а c — количество столбцов матрицы M (равное 3).

Зададим матрицы преобразований, где D — матрица перемещений, S — масштабирования, R_x — вращения (относительно оси X):

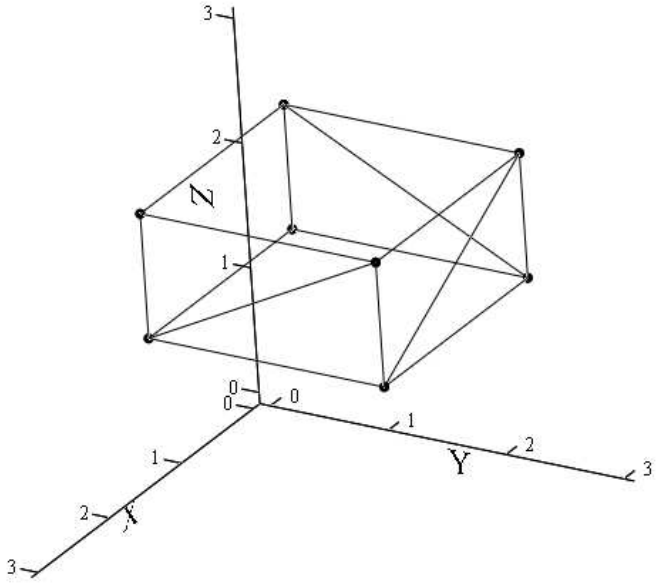
$$D := \begin{pmatrix} 1 & 0 & 0 & D_X \\ 0 & 1 & 0 & D_y \\ 0 & 0 & 1 & D_Z \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad S := \begin{pmatrix} S_X & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_Z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad R_X := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

где:

$D_X := 2 \qquad D_y := -1 \qquad D_Z := 3$

$S_X := 1.5 \qquad S_y := 1.5 \qquad S_Z := 2.5$

$\alpha := -\frac{\pi}{4}$



$(M^{\langle 0 \rangle}, M^{\langle 1 \rangle}, M^{\langle 2 \rangle})$

Рисунок П.3 — Исходное изображение (прямоугольный параллелепипед).

Преобразуем координаты фигуры в однородные, для этого составим программный фрагмент:

$$
\begin{array}{l}
M' := \left| \begin{array}{l}
\text{for } i \in 0 .. r - 1 \\
\quad \left| \begin{array}{l}
\text{for } j \in 0 .. c - 1 \\
\quad m''_{j,0} \leftarrow M_{i,j} \\
m''_{c,0} \leftarrow 1 \\
m'_i \leftarrow m''
\end{array} \right. \\
M' \leftarrow m'
\end{array} \right.
\end{array}
$$

В итоге получаем вектор векторов *M′* (одномерный массив, каждый элемент которого является одномерным массивом [4 x 1]) с однородными координатами каждой точки:

M'^{T} =

	0	1	2	3	4	5	6	7	8
0	[4, 1]	[4, 1]	[4, 1]	[4, 1]	[4, 1]	[4, 1]	[4, 1]	[4, 1]	...

Здесь для удобства вывода результата M'^{T} — транспонированная матрица.

Далее создадим программный фрагмент для преобразования фигуры в однородных координатах, который заключается в перемножении матрицы преобразования (параметр *Tr*) на исходную матрицу *M′* с однородными координатами:

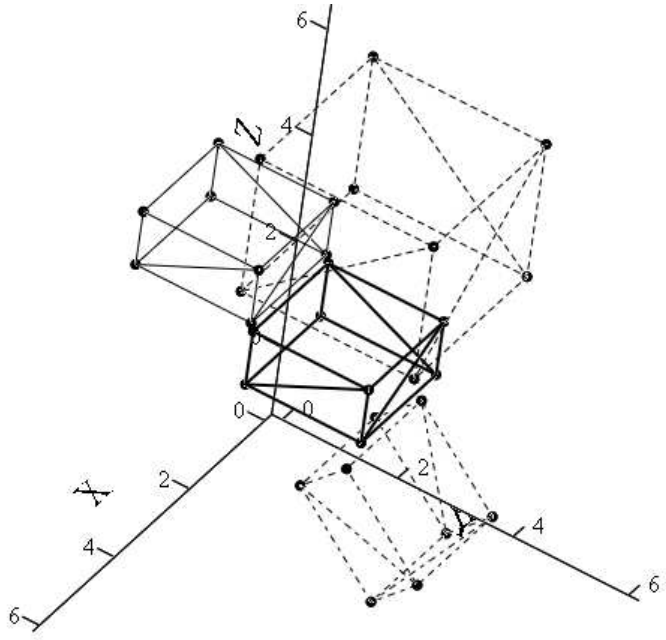
$$M'_T(Tr) := \left| \begin{array}{l} \text{for } i \in 0..r-1 \\ \quad \left| \begin{array}{l} m'' \leftarrow Tr \cdot M'_i \\ m'_i \leftarrow m'' \end{array} \right. \\ M'_T \leftarrow m' \end{array} \right.$$

Правильность преобразований можно проверить, задав в качестве параметра 1 (единицу), таким образом, получив исходную матрицу.

Теперь необходимо преобразовать матрицу однородных координат в матрицу декартовых (избавиться от 1), для этого составим программный фрагмент:

$$M_T(Tr) := \left| \begin{array}{l} \text{for } i \in 0..r-1 \\ \quad \left| \begin{array}{l} \text{for } j \in 0..c-1 \\ \quad m'_{i,j} \leftarrow \left(M'_T(Tr)_i\right)_j \\ m \leftarrow m' \end{array} \right. \\ M_T \leftarrow m \end{array} \right.$$

Рассчитав все координаты, изобразим полученные преобразования в пространстве (рисунок П.4).



$$\left(M^{\langle 0\rangle}, M^{\langle 1\rangle}, M^{\langle 2\rangle}\right), \left(M_{T(D)}{}^{\langle 0\rangle}, M_{T(D)}{}^{\langle 1\rangle}, M_{T(D)}{}^{\langle 2\rangle}\right), \left(M_{T(S)}{}^{\langle 0\rangle}, M_{T(S)}{}^{\langle 1\rangle}, M_{T(S)}{}^{\langle 2\rangle}\right), \left(M_T\left(R_x\right)^{\langle 0\rangle}, M_T\left(R_x\right)^{\langle 1\rangle}, M_T\left(R_x\right)^{\langle 2\rangle}\right)$$

Рисунок П.4 — Геометрические преобразования 3-D фигуры.

Раздел 3. Алгоритмы работы с геометрическими объектами

Глава 5. Отсечение геометрических примитивов

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

1 Алгоритм Сазэрленда-Коэна отсечения прямоугольной областью

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

$$x_l \le x \le x_r,\ y_b \le y \le y_t. \tag{32}$$

Пусть концы отрезка заданы точками (x_1, y_1) и (x_2, y_2). Первый шаг алгоритма нацелен на то, чтобы выявить полностью видимые и полностью невидимые отрезки. Отрезок целиком принадлежит выделяемой (клиппируемой) области, если оба его конца удовлетворяют условиям (32).

Отрезок полностью невидим, если оба его конца лежат

- справа от ребра $r(x_1 > x_r, x_2 > x_r)$;
- слева от ребра $l(x_1 < x_l, x_2 < x_l)$;
- снизу от ребра $b(y_1 < y_b, y_2 < y_b)$;
- сверху от ребра $t(y_1 > y_t, y_2 > y_t)$.

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

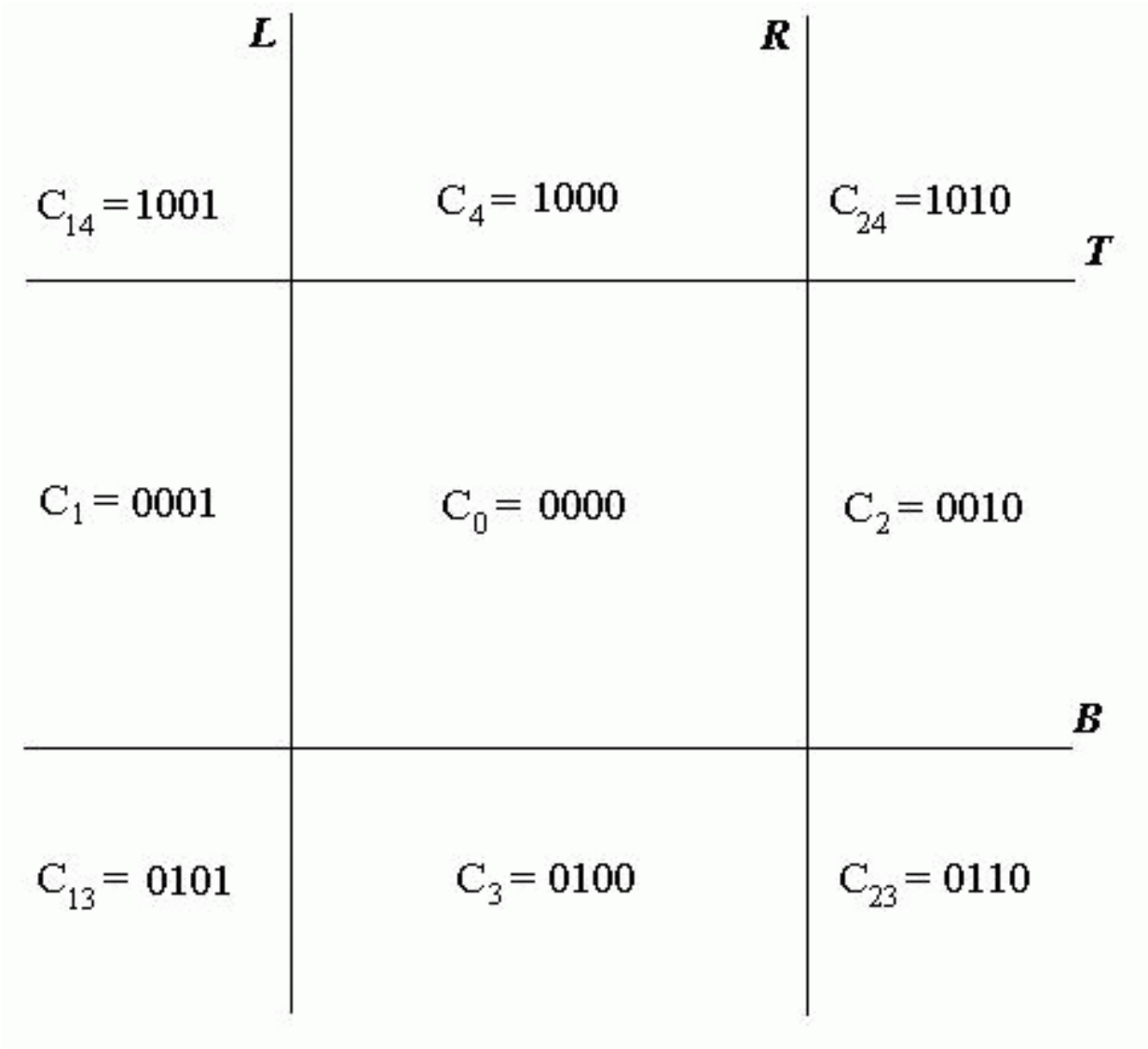


Рисунок 43 — Коды Коэна-Сазерленда для областей.

Для выполнения анализа полной видимости или невидимости отрезка А.Сазерленд и Д.Коэн предложили следующий алгоритм. Прямые, которым принадлежат рёбра прямоугольника, разбивают плоскость на девять областей, каждой из которых присваивается четырёхразрядный код. Каждый бит этого кода "отвечает" за одну из прямых: 1-й (старший) бит — за прямую *t*, 2-й — за прямую *b*, 3-й — за *r*, 4-й — за *l*. Если в коде области какой-либо бит установлен в 1, то это означает, что она отделена от окна соответствующей прямой. Схема идентификации областей приведена на рисунке 43.

Концевым точкам отрезков сцены теперь можно присвоить коды в зависимости от расположения точек. Ясно, что если коды обоих концов отрезка равны нулю, то отрезок полностью лежит внутри окна. Для дальнейшего анализа воспользуемся операцией логического умножения кодов (поразрядное логическое "И"). Тогда таблица истинности для кодов, согласно схеме на рисунке 43, будет выглядеть следующим образом:

Таблица 1 — Значения истинности для логического умножения кодов областей

	C_1	C_2	C_3	C_4	C_{13}	C_{14}	C_{23}	C_{24}
C_1	**T**rue	**F**alse	**F**alse	**F**alse	**T**rue	**T**rue	**F**alse	**F**alse
C_2	**F**alse	**T**rue	**F**alse	**F**alse	**F**alse	**F**alse	**T**rue	**T**rue
C_3	**F**alse	**F**alse	**T**rue	**F**alse	**T**rue	**F**alse	**T**rue	**F**alse
C_4	**F**alse	**F**alse	**F**alse	**T**rue	**F**alse	**T**rue	**F**alse	**T**rue
C_{13}	**T**rue	**F**alse	**T**rue	**F**alse	**T**rue	**T**rue	**T**rue	**F**alse
C_{14}	**T**rue	**F**alse	**F**alse	**T**rue	**T**rue	**T**rue	**F**alse	**T**rue
C_{23}	**F**alse	**T**rue	**T**rue	**F**alse	**T**rue	**F**alse	**T**rue	**T**rue
C_{24}	**F**alse	**T**rue	**F**alse	**T**rue	**F**alse	**T**rue	**T**rue	**T**rue

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

В самом общем случае существуют две точки пересечения отрезка с ребрами, и эти две точки принимаются за новые концевые точки изображаемого отрезка. Но сначала можно выделить некоторые более простые частные случаи, поиск пересечений для которых является более эффективным. Прежде всего, это горизонтальные и вертикальные отрезки, для которых поиск точки пересечения

тривиален. Далее, если код одного из концов отрезка равен нулю, то существует только одно пересечение этого отрезка с ребром (или с двумя ребрами, если отрезок проходит через угловую точку окна). На рисунке 44 приведена общая блок-схема алгоритма отсечения для одного произвольно направленного отрезка.

В блок-схеме используются следующие соглашения и обозначения:

- входными данными являются точки $\overrightarrow{r_1} = (x_1, y_1), \overrightarrow{r_2} = (x_2, y_2)$, массив координат окна $w = \{L, R, B, T\}$; $\vec{l} = (x_2 - x_1, y_2 - y_1) \equiv (l_x, l_y)$;
- на выходе получаем новые концевые точки $\overrightarrow{r_{10}} = (x_{10}, y_{10}), \overrightarrow{r_{20}} = (x_{20}, y_{20})$, а также значение переменной *IsVisible* (0 — отрезок невидимый);
- используются следующие вспомогательные функции:

  GetCode(r) — определение кода точки;

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

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

  C1, *C2* — коды точек *r1*, *r2*.

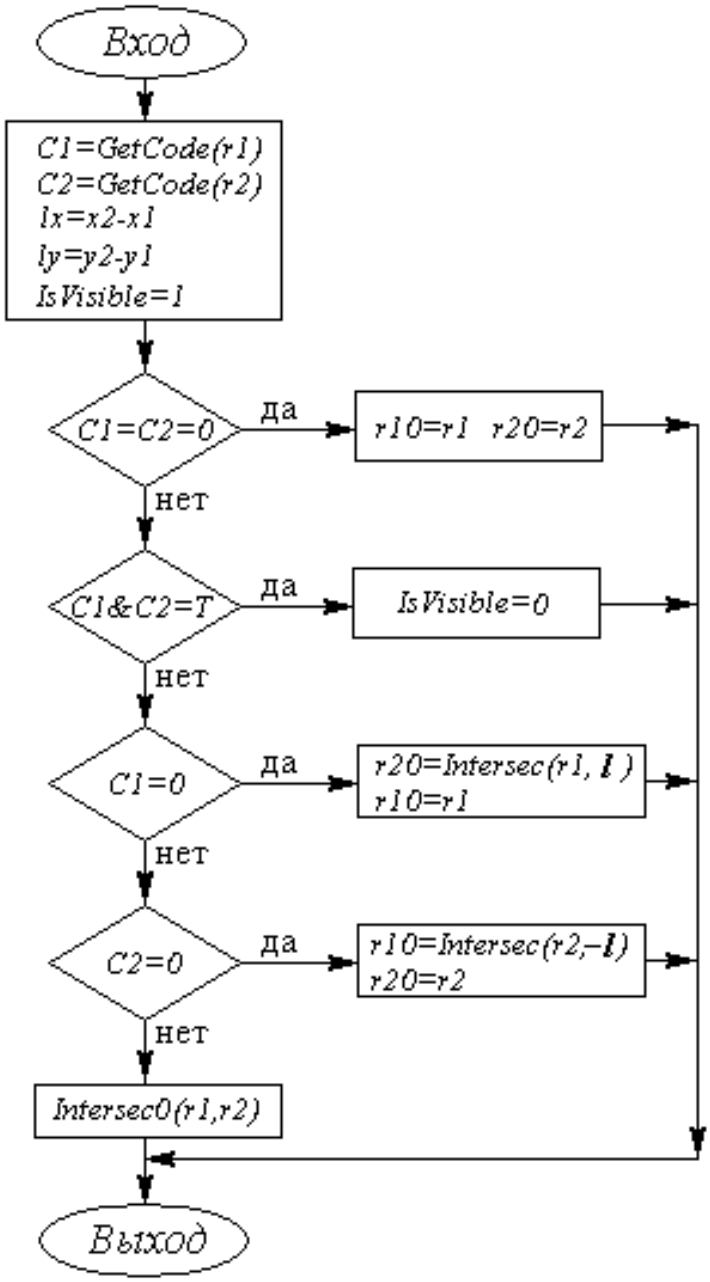


Рисунок 44 — Графическая схема алгоритма отсечения невидимых линий Сазерленда-Коэна.

В приведённом алгоритме теперь остаётся только детализовать функции *Intersec0* и *Intersec*, эффективность работы которых является ключевым моментом. Блок-схемы алгоритмов этих функций изображены на рисунках 45 и 46.

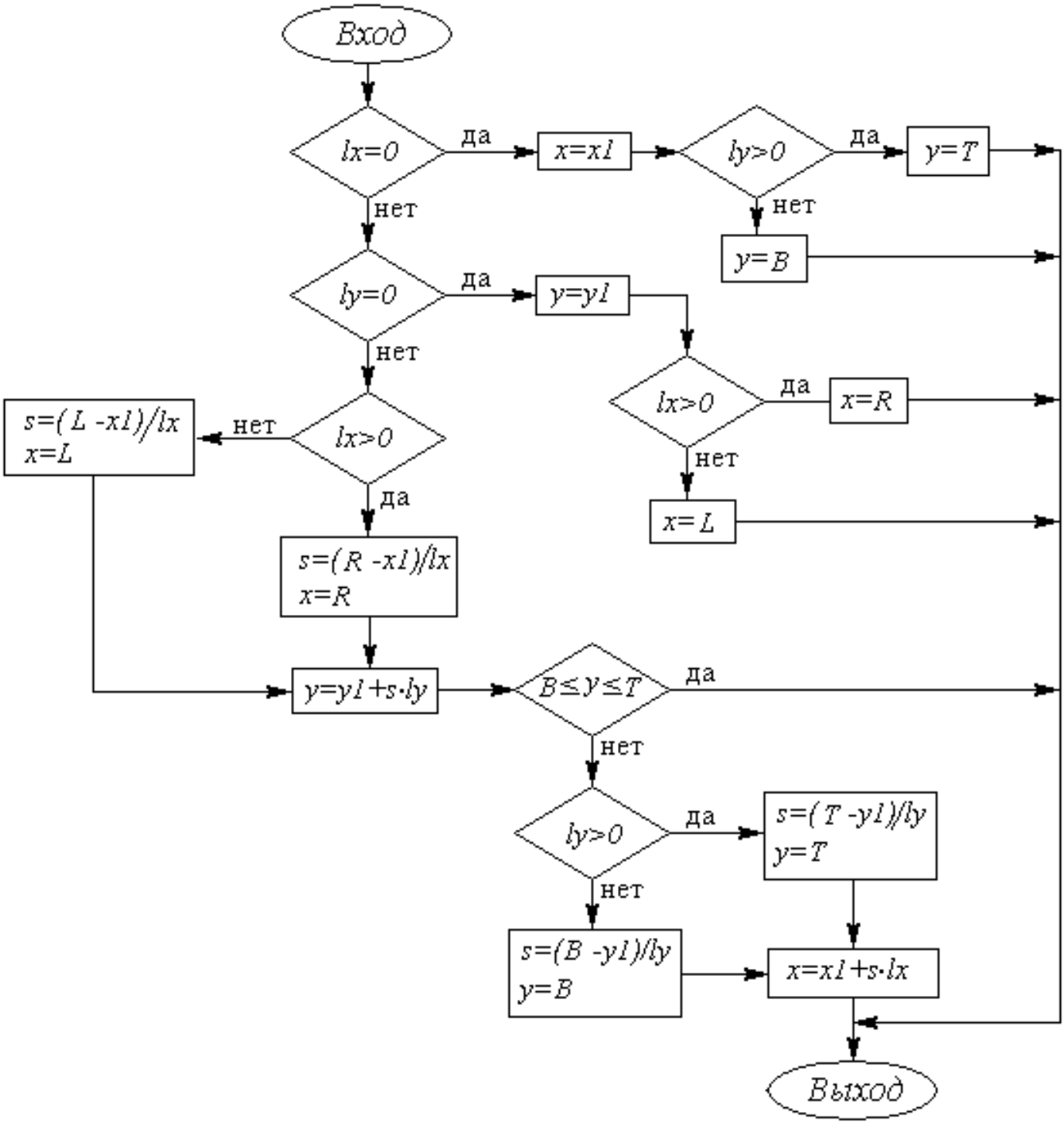


Рисунок 45 — Графическая схема алгоритма работы функции *Intersec*.

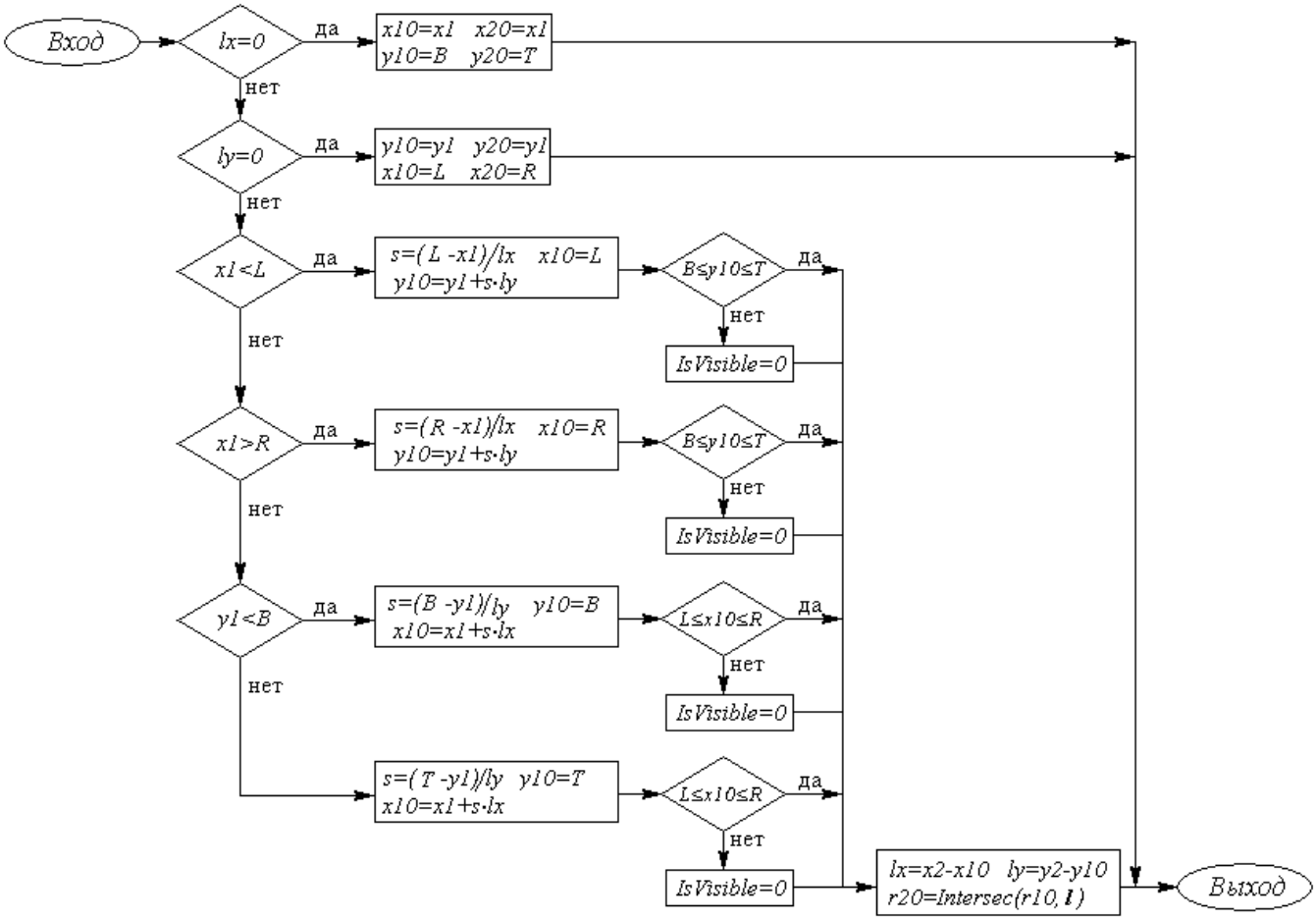


Рисунок 46 — Графическая схема алгоритма работы функции *Intersec0*.

Предложенная реализация алгоритма отсечения не является единственной в своем роде. Существуют и другие подходы, в частности использование метода деления отрезка пополам для поиска точек пересечения и выявления видимой части отрезка. Такой вариант алгоритма был предложен Сазерлендом и Спрулом[29] для аппаратной реализации, но он может быть реализован и на программном уровне, хотя при этом эффективность его будет ниже, чем у предыдущего. В нём нет прямого вычисления координат новой точки по явным алгебраическим соотношениям. Поиск осуществляется итерационным методом, в котором на каждом шаге для отрезка, "подозреваемого" в частичной видимости, находится его средняя точка, определяется её код, затем из двух отрезков оставляются либо оба (если они оба не окажутся полностью невидимыми), либо только один, после чего операции продолжаются с новыми отрезками. Ситуация, когда дальнейшему дроблению подвергаются сразу два вновь полученных отрезка, может возникнуть только на первом итерационном шаге в тех случаях, когда исходный отрезок имеет

[29] Боб Спроул (Robert Fletcher "Bob" Sproull) — студент Айвена Сазерленда. В дальнейшем разработчик программного обеспечения, директор Oracle Labs в Берлингтоне, штат Массачусетс.

две точки пересечения с границами окна. На последующих итерационных шагах количество анализируемых отрезков уже не будет увеличиваться. Процесс продолжается до тех пор, пока длина очередного отрезка не станет меньше наперед заданной точности. После этого найденная точка проверяется на предмет пересечения со стороной окна. На рисунке 47 приведены три варианта отрезков, предварительный анализ которых не классифицирует их как полностью невидимые, и показан первый итерационный шаг. Отрезок *a* после первого деления даёт два частично видимых отрезка, после чего ищутся две точки пересечения. В остальных случаях остается лишь один из двух отрезков, причём в случае отрезка *c* точка пересечения со стороной окна отсутствует, т.е. обнаруживается полная невидимость отрезка. По сути дела общий алгоритм, показанный на рисунке 45, сохраняется, изменяется лишь метод поиска точки пересечения.

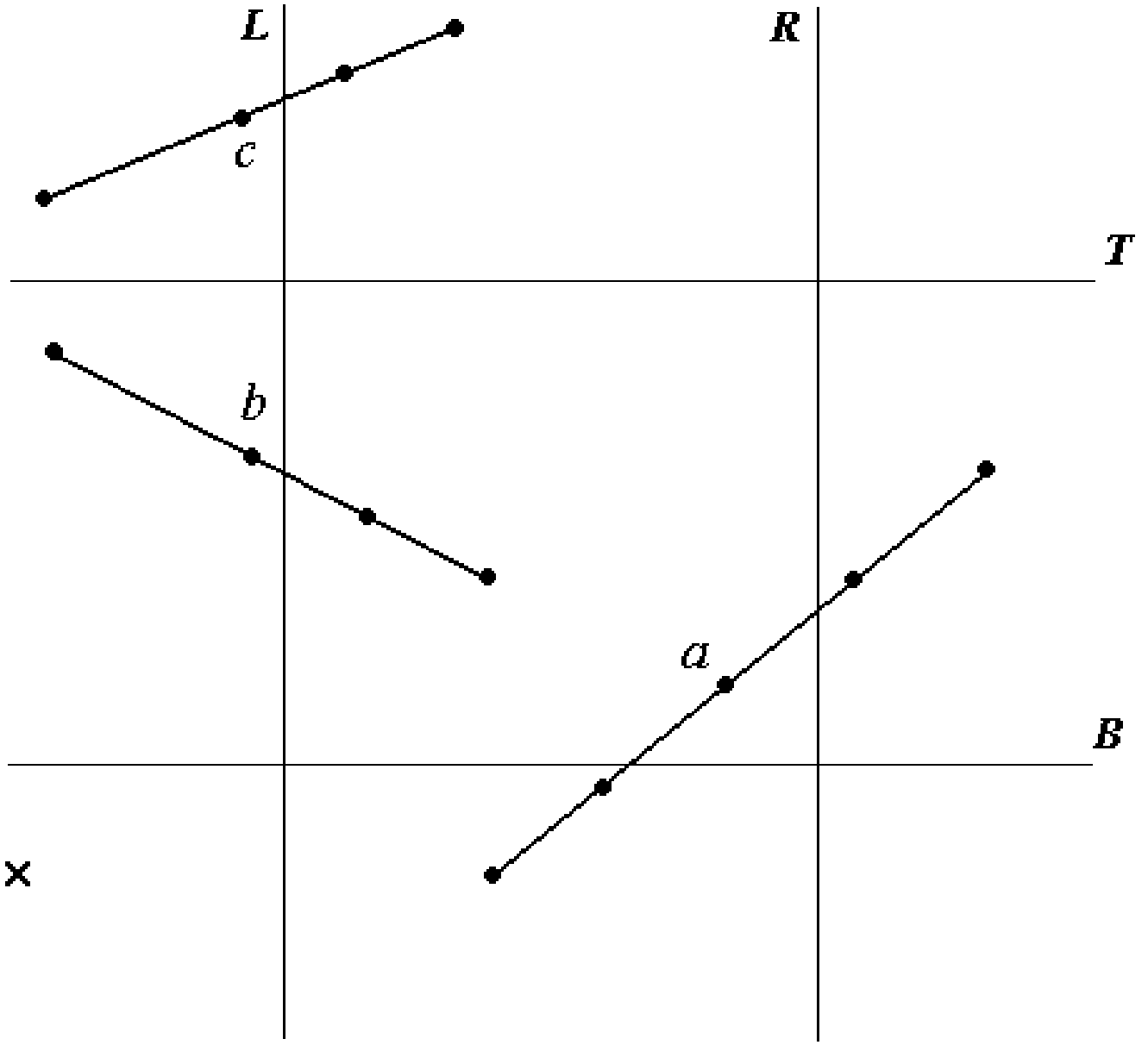


Рисунок 47 — Произвольное расположение отрезков.

2 Отсечение выпуклым многоугольником

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

Рассмотрим задачу о пересечении отрезка с концами $\overrightarrow{r_1} = (x_1, y_1), \overrightarrow{r_2} = (x_2, y_2)$ с выпуклым многоугольником, заданным списком рёбер. Ребро может быть задано в виде пары точек из множества вершин многоугольника $R = (\vec{p}, \vec{q})$, $\vec{p} = (p_x, p_y)$, $\vec{q} = (q_x, q_y)$ (рисунок 48). То обстоятельство, что многоугольник выпуклый, является очень существенным: это позволяет использовать достаточно простой алгоритм, использующий внутренние нормали к его сторонам. Под внутренней нормалью понимается вектор, перпендикулярный стороне и направленный внутрь многоугольника. Как и в предыдущем алгоритме, воспользуемся параметрическим уравнением прямой, проходящей через концы отрезка: $\vec{r} = \overrightarrow{r_1} + s(\overrightarrow{r_2} - \overrightarrow{r_1})$. Если при некотором значении параметра s_0 эта прямая пересекается с прямой, проходящей через точки $\vec{p}, \vec{q}$, то вектор, соединяющий произвольную точку ребра с точкой $\overrightarrow{r_0} = \overrightarrow{r_1} + s_0(\overrightarrow{r_2} - \overrightarrow{r_1})$, будет перпендикулярен вектору нормали. Следовательно, скалярное произведение векторов $\vec{n}$ и $\vec{p} - \overrightarrow{r_0}$ будет равно нулю. Отсюда путём несложных выкладок получаем $s_0 = \frac{(\vec{p} - \overrightarrow{r_1}) \cdot \vec{n}}{(\overrightarrow{r_2} - \overrightarrow{r_1}) \cdot \vec{n}}$.

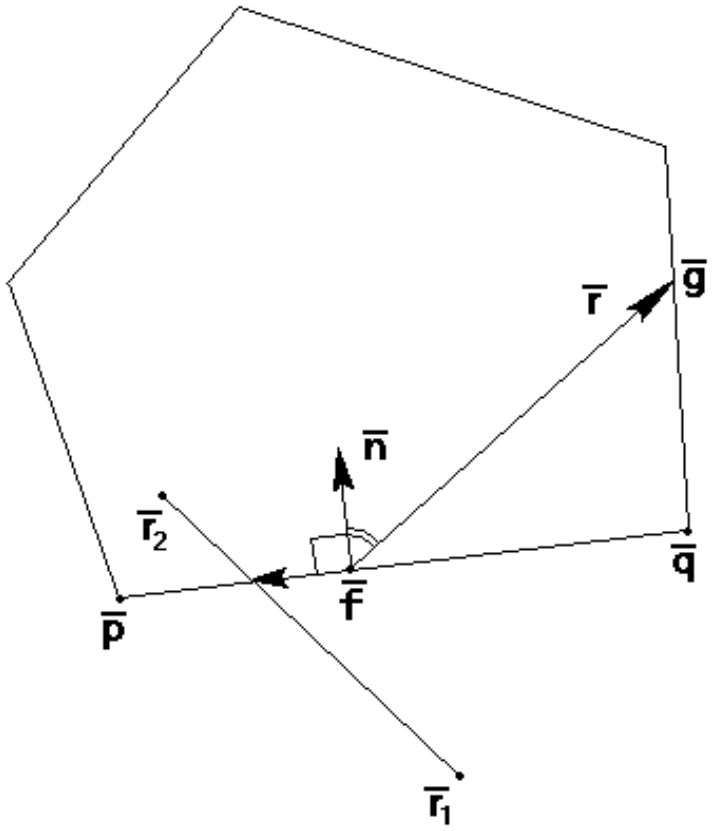


Рисунок 48 — Пересечение отрезка многоугольником.

Конечно, использование этой формулы предполагает, что отрезок не параллелен стороне многоугольника. Найденная точка принадлежит отрезку при условии $0 \le s_0 \le 1$. Условие принадлежности этой точки ребру многоугольника также можно выразить через скалярное произведение, так как векторы $\vec{p} - \overrightarrow{r_0}$ и $\overrightarrow{r_0} - \vec{q}$ в этом случае должны быть одинаково направленными, т.е. $(\vec{p} - \overrightarrow{r_0}) \cdot (\overrightarrow{r_0} - \vec{q}) \ge 0$.

Для каждого отрезка возможны три случая взаимного расположения с многоугольником:

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

В каждом из этих вариантов для нахождения пересечения отрезка с окном необходимо уметь определять принадлежность точки выпуклому многоугольнику. Из рисунка 48 видно, что если для любой точки $\vec{g}$, принадлежащей многоугольнику (или его границе), и произвольной точки ребра $\vec{f}$ построить вектор $\vec{m} = \vec{g} - \vec{f}$, то выполняется условие $\vec{m} \cdot \vec{n} \ge 0$, поскольку угол между векторами не может превышать 90° Таким образом, если данное условие выполняется для всех рёбер многоугольника, то точка является внутренней.

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

Далее выполняется цикл по всем рёбрам многоугольника с целью нахождения точек пересечения. Для каждого ребра перед началом поиска пересечения необходимо проверить, не параллельно ли оно с отрезком. Если это так, то можно вычислить расстояние от одного из концов отрезка до прямой, проходящей через ребро $d = (\overrightarrow{r_1} - \vec{p}) \cdot \vec{n}$. При $d = 0$ отрезок лежит на прямой, и остается определить взаимное расположение концов отрезка и концов ребра, что можно сделать простым покоординатным сравнением. При $d \neq 0$ отрезок не имеет общих точек с данным ребром.

3 Клиппирование многоугольников

От задачи отсечения отрезков можно перейти к более сложной: клиппирование произвольных многоугольников. Если многоугольник невыпуклый, то в результате пересечения даже с прямоугольным окном может получиться несколько несвязанных между собой фигур, как это показано на рисунке 49.

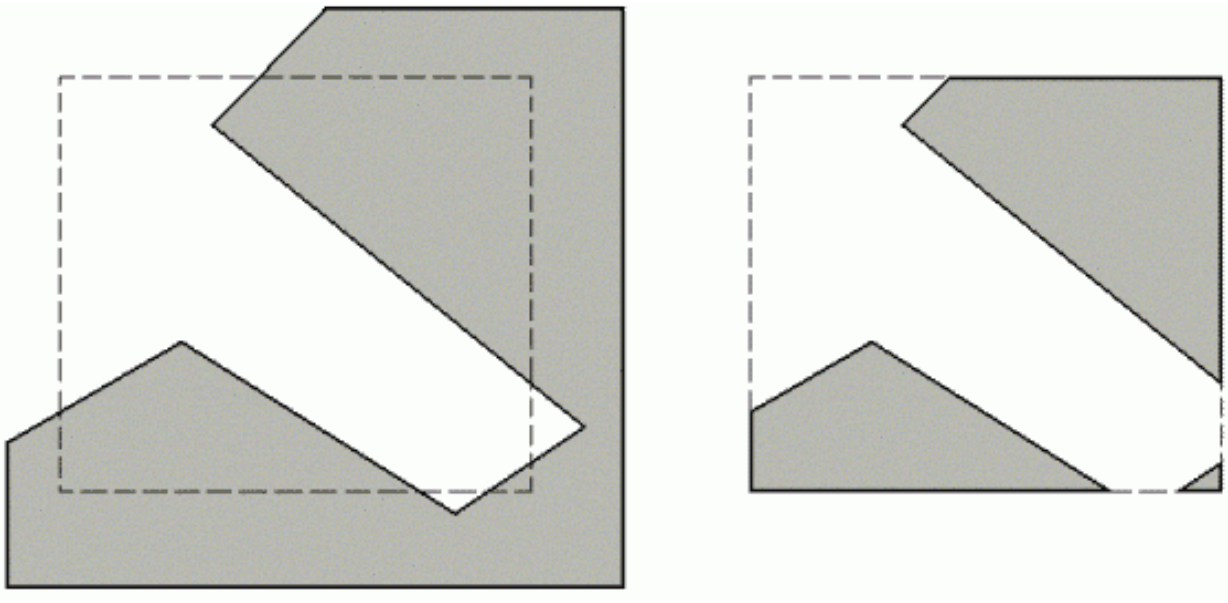


Рисунок 49 — Отсечение невыпуклого многоугольника.

Когда стоит задача штрихования замкнутой области одновременно с задачей отсечения, то важно правильно определить принадлежность вновь полученных фигур внутренней или внешней части исходного многоугольника.

Пусть исходный многоугольник задан упорядоченным списком вершин $\{\overrightarrow{p_1}, \overrightarrow{p_2}, \dots, \overrightarrow{p_n}\}$, соответствующих рёбрам $(\overrightarrow{p_1}, \overrightarrow{p_2}), (\overrightarrow{p_2}, \overrightarrow{p_3}), \dots, (\overrightarrow{p_{n-1}}, \overrightarrow{p_n}), (\overrightarrow{p_n}, \overrightarrow{p_1})$. Для отсечения такого многоугольника прямоугольным окном можно применить алгоритм, предложенный Сазерлендом и Ходжменом. Идея его заключается в последовательном отсечении части многоугольника прямыми, соответствующими сторонам окна. Результатом его работы является упорядоченный список вершин, лежащих в видимой части окна. На каждом шаге алгоритма образуется некоторая промежуточная фигура, также представленная упорядоченным списком вершин и рёбер. Пример таких последовательных отсечений показан на рисунке 50. В процессе отсечения последовательно обходится список вершин, причём каждая очередная точка за исключением первой рассматривается как конечная точка ребра, начальной точкой которого является предшествующая точка из списка. Порядок, в котором рассматриваются стороны окна, не имеет значения. В процессе обхода формируется список новых вершин многоугольника.

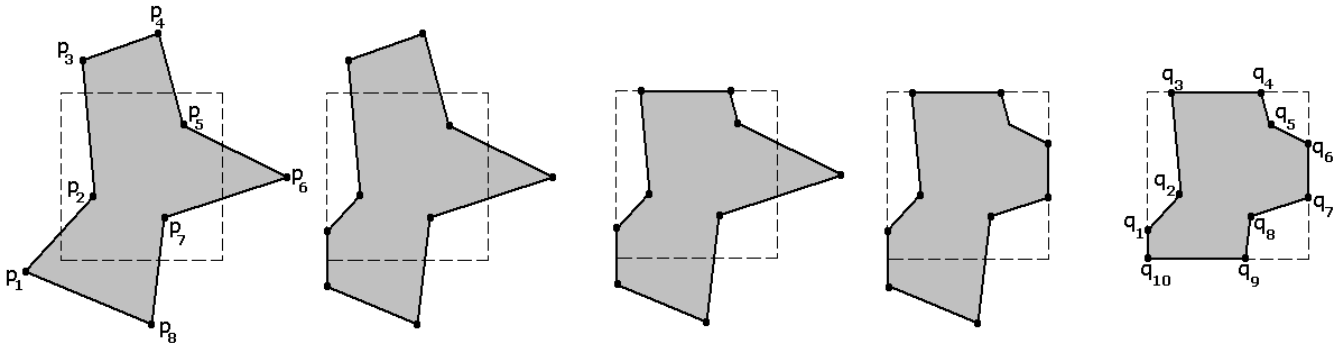


Рисунок 50 — Последовательные шаги клиппирования произвольного многоугольника.

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

Для анализируемого ребра возможны четыре случая расположения относительно окна.

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

В этом алгоритме постоянно приходится определять видимость точки по отношению к конкретному ребру отсекающего окна. Окно также можно задать в виде упорядоченного списка вершин. Если обход вершин окна осуществляется по часовой стрелке, то его внутренняя область будет расположена по правую сторону от границы. При этом расположение точки $\vec{p}$ относительно прямой, которой принадлежит ребро, можно устанавливать различными способами:

1. Выбирается начальная точка $\vec{s}$ данного ребра и строится вектор $\vec{r} = \vec{p} - \vec{s}$ и вектор внутренней нормали n к границе (ребру) окна. Вычисляется скалярное произведение $d = \vec{r} \cdot \vec{n}$. Если $d \geq 0$, то точка $\vec{p}$ является видимой.
2. Строится пространственный вектор $\vec{r}$ (третью координату можно положить равной нулю). Вектор $\vec{l}$ (также пространственный, лежащий в той же плоскости) направлен вдоль ребра (с учетом направления обхода). Вычисляется векторное произведение $\vec{v} = [\vec{r} \times \vec{l}]$. Если координата z у вектора $\vec{v}$ положительна, то точка $\vec{p}$ лежит справа от ребра (является видимой).
3. Выписывается каноническое уравнение прямой, проходящей через ребро: $f(x, y) + b \cdot y + c = 0$.
4. Для произвольной внутренней точки окна (x_0, y_0) вычисляется значение $d = f(x_0, y_0)$, а также для точки $\vec{p} = (x_1, y_1)$ вычисляется $d_1 = f(x_1, y_1)$. Если числа d и d_1 имеют одинаковый знак, то точка $\vec{p}$ является видимой.

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

Глава 6. Удаление невидимых поверхностей и линий

Задача удаления невидимых линий и поверхностей является одной из наиболее интересных и сложных в компьютерной графике. Алгоритмы удаления заключаются в определении линий рёбер, поверхностей или объёмов, которые видимы или невидимы для наблюдателя, находящегося в заданной точке пространства.

Необходимость удаления невидимых линий, рёбер, поверхностей или объёмов проиллюстрирована на рисунке 51. Рисунок наглядно демонстрирует, что изображение без удаления невидимых линий воспринимается неоднозначно.

Рисунок 51 — Неоднозначность восприятия изображения куба.

Сложность задачи удаления невидимых линий и поверхностей привела к появлению большого числа различных способов её решения. Многие из них ориентированы на специализированные приложения. Единого (общего) решения этой задачи, годного для различных случаев не существует: для каждого случая выбирается наиболее подходящий метод. Например, для моделирования процессов в реальном времени требуются быстрые алгоритмы, в то время как для формирования сложного реалистического изображения, в котором представлены тени, прозрачность и фактура, учитывающие эффекты отражения и преломления цвета в мельчайших оттенках, фактор времени выполнения уже не так существенен. Подобные алгоритмы работают медленно, и зачастую на вычисления требуется несколько минут или даже часов. Существует тесная взаимосвязь между скоростью работы алгоритма и детальностью его результата. Ни один из алгоритмов не может достигнуть хороших оценок для этих двух показателей одновременно. По мере создания всё более быстрых алгоритмов можно строить всё более детальные изображения. Реальные задачи, однако, всегда будут требовать учёта еще большего количества деталей.

Все алгоритмы такого рода, так или иначе, включают в себя сортировку, причём главная сортировка ведётся по геометрическому расстоянию от тела, поверхности, ребра или точки до точки наблюдения или картинной плоскости.

Основная идея, положенная в основу сортировки по расстоянию, заключается в том, что чем дальше расположен объект от точки наблюдения, тем больше вероятность, что он будет полностью или частично заслонен одним из объектов, более близких к точке наблюдения. После определения расстояний или приоритетов по глубине остаётся провести сортировку по горизонтали и по вертикали, чтобы выяснить, будет ли рассматриваемый объект действительно заслонен объектом, расположенным ближе к точке наблюдения. Эффективность любого алгоритма удаления в значительной мере зависит от эффективности процесса сортировки.

Алгоритмы удаления невидимых линий или поверхностей можно классифицировать по способу выбора системы координат или пространства, в котором они работают. Алгоритмы, работающие в объектном пространстве[30], имеют дело с мировой системой координат[31], в которой описаны эти объекты. При этом получаются весьма точные результаты, ограниченные, вообще говоря, лишь погрешностью вычислений. Полученные изображения можно свободно масштабировать. Алгоритмы, работающие в объектном пространстве, особенно полезны в тех приложениях, где необходима высокая точность. Алгоритмы же, работающие в пространстве изображения, имеют дело с системой координат того экрана, на котором объекты визуализируются. При этом точность вычислений ограничена разрешающей способностью экрана.

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

---

[30] Каждое множество объектов определяет свойственное ему объектное пространство — систему пространственных отношений между объектами. Объектное пространство позволяет однозначно идентифицировать объекты, указывая для каждого из них определенное место, отличное от мест других объектов.

Именно в объектном координатном пространстве происходит построение модели (объекта).

Начало объектного пространства нужно располагать так, чтобы было удобней создавать объект. Иногда начало объектного пространства совпадает с центром масс модели.

[31] Единицы измерения, в которых задаются объекты, зависят от их природы: изменение температуры, например, можно отображать в градусах за час, перемещение тела в пространстве — в километрах в секунду, и т. д. Эти прикладные (или ориентированные на пользователя) координаты позволяют задавать объекты в двумерном или трехмерном мире пользователя, и их принято называть мировыми координатами.

1 Удаление нелицевых граней многогранника

1.1 Алгоритм Робертса

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

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

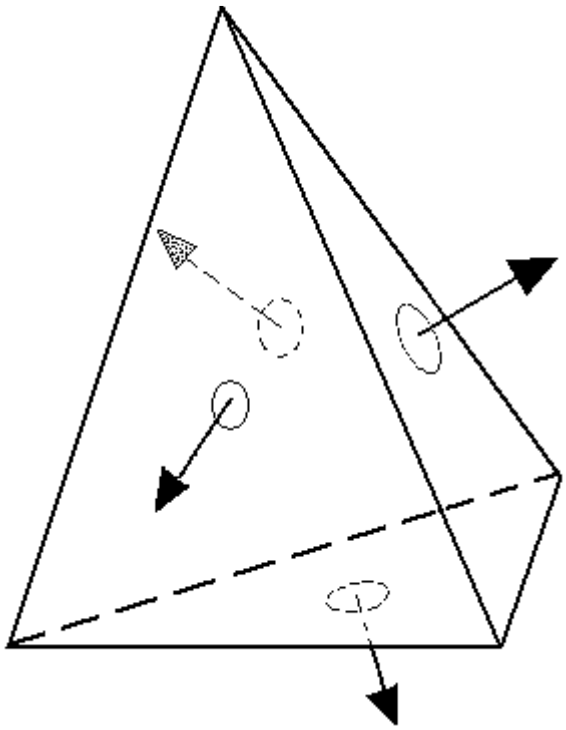


Рисунок 52 — Внешние нормали тетраэдра.

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

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

Принадлежность точки плоскости можно установить с помощью скалярного произведения, т.е. если $\vec{P} \cdot \vec{r} = 0$, то точка принадлежит плоскости, если же нет, то знак произведения показывает, по какую сторону от плоскости эта точка находится. В алгоритме Робертса плоскости строятся таким образом, что внутренние точки многогранника лежат в положительной полуплоскости. Это означает, что вектор (A, B, C) является внешней нормалью к многограннику (рисунок 52). Из векторов плоскостей строится прямоугольная матрица порядка [4 x n], которая называется обобщенной матрицей описания многогранника:

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

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

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

Из обобщенной матрицы можно получить информацию о том, какие грани многогранника пересекаются в вершинах. Действительно, если вершина $\vec{v} = (x, y, z, 1)$ принадлежит граням $\overrightarrow{P_1}, \overrightarrow{P_2}, \overrightarrow{P_3}$, то она удовлетворяет уравнениям:

$$\left.\begin{aligned} \vec{v} \cdot \overrightarrow{P_1} &= 0 \\ \vec{v} \cdot \overrightarrow{P_2} &= 0 \\ \vec{v} \cdot \overrightarrow{P_3} &= 0 \\ \vec{v} \cdot \overrightarrow{e_4} &= 0 \end{aligned}\right\}, \text{ где } \overrightarrow{e_4} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

Эту систему можно записать в матричном виде:

$$\vec{v} \cdot Q = \overrightarrow{e_4},$$

где Q — матрица, составленная из векторов-столбцов $\overrightarrow{P_1}, \overrightarrow{P_2}, \overrightarrow{P_3}, \overrightarrow{e_4}$. Значит, координаты вершины определяются соотношением:

$$\vec{v} = \overrightarrow{e_4} \cdot Q^{-1},$$

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

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

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

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

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

а прямая, идущая в точку, соответствующую параметру *t*, — уравнением:

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

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

$$\vec{q_i} = \left(\vec{v} \cdot \vec{P_i}\right) + t\left(\vec{d} \cdot \vec{P_i}\right) + \tau\left(\vec{g} \cdot \vec{P_i}\right) > 0.$$

Эти условия должны выполняться для всех плоскостей.

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

Так как объём вычислений растёт с увеличением числа многоугольников, то желательно по мере возможности сокращать их число, т.е. если аппроксимируется некоторая поверхность многогранником, то в качестве граней можно использовать не треугольники, а более сложные многоугольники. При этом, разумеется, встает

проблема, как построить такой многоугольник, чтобы он мало отклонялся от плоской фигуры.

Приведём эффективную реализацию алгоритма Робертса.

Этот алгоритм делится на три этапа:

1) На первом этапе каждое тело анализируется индивидуально с целью удаления нелицевых плоскостей.
2) На втором этапе проверяется экранирование оставшихся в каждом теле ребер всеми другими телами с целью обнаружения их невидимых отрезков.
3) 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 рёбра — из отдельных вершин. Все вершины, ребра и грани связаны с конкретным телом.

```
Удаление нелицевых плоскостей
    Для каждого тела в сцене:
        Сформировать многоугольники граней и рёбра, исходя из списка вершин тела.

        Вычислить уравнение плоскости для каждой полигональной грани тела.

        Проверить знак уравнения плоскости:

            Взять любую точку внутри тела, например, усреднив координаты его вершин.

            Вычислить скалярное произведение уравнения плоскости и точки внутри тела.

            Если это скалярное произведение < 0, то изменить знак уравнения этой плоскости.

        Сформировать матрицу тела.

        Умножить её слева на матрицу, обратную матрице видового
        преобразования, включающего перспективу.

        Вычислить и запомнить габариты прямоугольной объемлющей
        оболочки преобразованного объёма: Xmax, Xmin, Ymax, Ymin.

        Определить нелицевые плоскости:

            Вычислить скалярное произведение пробной точки, лежащей
            в бесконечности, на преобразованную матрицу тела.

            Если это скалярное произведение < 0, то плоскость невидима.

            Удалить весь многоугольник, лежащий в этой плоскости.
            Это избавляет от необходимости отдельно рассматривать
            невидимые линии, образуемые пересечением пар невидимых
            плоскостей.

    Удаление из каждого тела тех рёбер, которые экранируются всеми
```

остальными телами в сцене:

Если задано только одно тело, то алгоритм завершается.

Сформировать приоритетный список этих тел:

Провести сортировку по z. Сортировка производится по максимальным значениям координаты z, вершин преобразованных тел. Первым в упорядоченном списке и обладающим наибольшим приоритетом будет то тело, у которого минимальное среди максимальных значений z. В используемой правой системе координат это тело будет самым удаленным от точки наблюдения, расположенной в бесконечности на оси z.

Для каждого тела из приоритетного списка:

Проверить экранирование всех лицевых ребер всеми другими телами сцены. Тело, ребра которого проверяются, называется пробным объектом, а тело, относительно которого в настоящий момент производится проверка, называется пробным телом. Естественно, что нужно проверять экранирование пробного объекта только теми пробными телами, у которых ниже приоритеты.

Провести проверки экранирования для прямоугольных объемлющих оболочек пробного объекта и пробного тела:

Если X_{min}(пробное тело) > X_{max}(пробный объект) или
X_{max}(пробное тело) < X_{min}(пробный объект) или
Y_{min}(пробное тело) > Y_{max}(пробный объект) или
Y_{max}(пробное тело) < Y_{min}(пробный объект),
то пробное тело не может экранировать ни одного ребра пробного объекта. Перейти к следующему пробному телу.
В противном случае:

Провести предварительные проверки протыкания, чтобы увидеть, не протыкается ли пробное тело пробным объектом и существует ли возможность частичного экранирования первого последним.

Сравнить максимальное значение z у пробного объекта с минимальным значением z у пробного тела.

Если Z_{max}(пробный объект) < Z_{min}(пробное тело), то протыкание невозможно. Перейти к следующему телу.
В противном случае:

Проверить видимое протыкание.

Если Z_{max}(пробный объект) > Z_{max}(пробное тело), то пробный объект может проткнуть переднюю грань пробного тела.

Установить флаг видимого протыкания для последующего использования. Занести проткнутое тело в список протыканий.

Если X_{max}(пробный объект) > X_{min}(пробное тело) или
X_{min}(пробный объект) < X_{max}(пробное тело),
то пробный объект может проткнуть бок пробного тела.

Установить флаг видимого протыкания для последующего использования. Занести тело в список протыканий.

Если Y_{max}(пробный объект) > Y_{min}(пробное тело) или
Y_{min}(пробный объект) < Y_{max}(пробное тело),
то пробный объект может проткнуть верх или низ пробного тела.

Установить флаг видимого протыкания для последующего использования. Занести проткнутое тело в список протыканий.

Если список протыканий пуст, то устанавливать флаг протыкания не надо.

Провести проверки экранирования ребер:

Вычислить s и d для ребра.
Вычислить p, q, w для каждой плоскости, несущей грань пробного тела.

Проверка полной видимости. Если ребро полностью видимо, то перейти к следующему ребру.

Сформировать уравнения $h_j = 0$ и решить их, объединяя попарно и включив в систему уравнения границ $t = 0$ и $t = 1$. Если установлен флаг видимого протыкания, то в систему надо включить и уравнение границы $a = 0$. Запомнить точки протыкания. В противном случае границу $a = 0$ не учитывать.

Для каждой пары (t, a), являющейся решением, проверить выполнение условий $0 <= t <= 1$, $a >= 0$ и $h_j > 0$ для всех других плоскостей. Если эти условия выполнены, то найти T_{max_min} и T_{min_max}.

Вычислить видимые участки отрезков и сохранить их для последующей проверки экранирования телами с более низкими приоритетами.

Определить видимые отрезки, связывающие точки протыкания:

Если флаг видимого протыкания не установлен, перейти к процедуре визуализации.

Если точек протыкания не обнаружено, перейти к процедуре визуализации.

Сформировать все возможные ребра, соединяющие точки протыкания, для пар тел, связанных отношением протыкания.

Проверить экранирование всех соединяющих ребер обоими телами, связанными отношением протыкания.

Проверить экранирование оставшихся соединяющих ребер всеми прочими телами сцены. Запомнить видимые отрезки.

Визуализировать оставшиеся видимые отрезки рёбер.

Заметим, что этот алгоритм можно реализовать и с обратным списком приоритетов.

Недостатки алгоритма Робертса :

- неспособность без привлечения других подходов реализовать падающие тени;
- невозможность передачи зеркальных эффектов и преломления света;
- строгая ориентация метода только на выпуклые многогранники.

Преимущество – одно:

- относительная простота.

1.2 Метод Z-буфера

Это один из простейших алгоритмов удаления невидимых поверхностей. Впервые он был предложен Эдвином Кэтмулом в 1975г. (по другим источникам, Вольфганг Штрассер описал эту идею в своей диссертации в 1974г.). Работает этот алгоритм в пространстве изображения. Идея Z-буфера является простым обобщением идеи о буфере кадра. Буфер кадра используется для запоминания атрибутов каждого пикселя в пространстве изображения, а Z-буфер предназначен для запоминания глубины (расстояния от картинной плоскости) каждого видимого пикселя в пространстве изображения. Поскольку достаточно распространенным является использование координатной плоскости *XOY* в качестве картинной плоскости, то глубина равна координате *z* точки, отсюда и название буфера. В процессе работы значение глубины каждого нового пикселя, который нужно занести в буфер кадра, сравнивается с глубиной того пикселя, который уже занесен в Z-буфер. Если это сравнение показывает, что новый пиксель расположен впереди пикселя, находящегося в буфере кадра, то новый пиксель заносится в этот буфер и, кроме того, производится корректировка Z-буфера новым значением глубины. Если же сравнение дает противоположный результат, то никаких действий не производится. По сути, алгоритм является поиском по *x* и *y* наибольшего значения функции $z(x, y)$.

Главное преимущество алгоритма — его простота. Кроме того, этот алгоритм решает задачу об удалении невидимых поверхностей и делает тривиальной визуализацию пересечений сложных поверхностей. Сцены могут быть любой сложности. Поскольку габариты пространства изображения фиксированы, оценка вычислительной трудоёмкости алгоритма не более чем линейна. Поскольку элементы сцены или картинки можно заносить в буфер кадра или в Z-буфер в произвольном порядке, их не нужно предварительно сортировать по приоритету глубины. Поэтому экономится вычислительное время, затрачиваемое на сортировку по глубине.

Основной недостаток алгоритма — большой объём требуемой памяти. В последнее время в связи с быстрым ростом возможностей вычислительной техники этот недостаток становится менее лимитирующим. Но в то время, когда алгоритм ещё только появился, приходилось изобретать способы создания буфера как можно большего объёма при имеющемся ресурсе памяти.

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

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

В целом алгоритм выглядит так:

1. Заполнить буфер кадра фоновым значением цвета.
2. Заполнить Z-буфер минимальным значением z (глубины).
3. Преобразовать изображаемые объекты в растровую форму в произвольном порядке.
4. Для каждого объекта:
   - 4.1 Для каждого пикселя (x, y) образа вычислить его глубину $z(x, y)$.
   - 4.2 Сравнить глубину $z(x, y)$ со значением глубины, хранящимся в Z-буфере в этой же позиции.
   - 4.3 Если $z(x, y) >$ Z-буфер (x, y), то занести атрибуты пикселя в буфер кадра и заменить Z-буфер (x, y) на $z(x, y)$. В противном случае никаких действий не производить.

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

$z(x, y) >$ Z-буфер (x, y) и $(x, y) = z$ сечения,

где z сечения — глубина искомого сечения.

Z-буфер представляет собой двумерный массив, каждый элемент которого соответствует пикселю на экране. Когда видеокарта рисует пиксель, его удалённость просчитывается и записывается в ячейку Z-буфера. Если пиксели двух рисуемых объектов перекрываются, то их значения глубины сравниваются, и рисуется тот, который ближе, а его значение удалённости сохраняется в буфер. Получаемое при этом графическое изображение носит название z-depth карта (карта глубины), представляющая собой полутоновое графическое изображение, каждый пиксель которого может принимать до 256 значений серого. По ним определяется удалённость от зрителя того или иного объекта трёхмерной сцены. Карта широко применяется в постобработке для придания объёмности и реалистичности и создаёт такие эффекты, как глубина резкости, атмосферная дымка и т.д. Также карта используется в 3D-пакетах для текстурирования, делая поверхность рельефной.

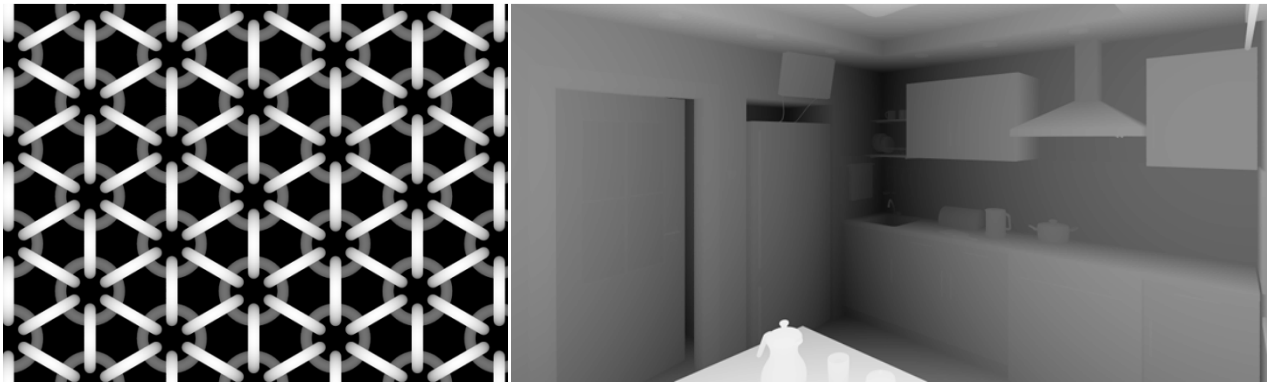


Рисунок 53 — Примеры карт глубины.

Разрядность буфера глубины оказывает сильное влияние на качество визуализации: использование 16-битного буфера может привести к геометрическим искажениям, например, эффекту «борьбы», если два объекта находятся близко друг к другу. 24, 32-разрядные буферы хорошо справляются со своей задачей. 8-битные почти никогда не используются из-за низкой точности.

В Z-буфере в его классическом виде разрядная сетка буфера недостаточно точна на близких расстояниях. Для решения этой проблемы применяется W-буфер, в котором применяется не удалённость, а обратная ей величина ($w = 1/z$). Что лучше применять — Z-буфер или W-буфер — зависит от программы.

На современных видеоадаптерах работа с Z-буфером отнимает немалую часть пропускной способности ОЗУ видеоадаптера. Для борьбы с этим применяют сжатие без потерь: сжатие/восстановление отнимает меньше ресурсов, чем обращение к памяти.

Если два объекта имеют близкую координату *z*, иногда, в зависимости от точки обзора, показывается то один, то другой, то оба полосатым узором. Это называется Z-конфликт. Чаще всего конфликты присущи спецэффектам, накладывающимся на основную текстуру (например, дырам от пуль).

Решаются Z-конфликты сдвигом одного объекта относительно другого на величину, превышающую погрешность Z-буфера.

Раздел 4. Проецирование. Алгоритмы преобразования геометрических объектов

Глава 7. Типы проекций

1 Основные типы проекций

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

Определенный таким образом класс проекций называют плоскими геометрическими проекциями, поскольку проецирование в этом случае производится на проекционную плоскость и в качестве проекторов используются прямые. Существуют и другие проекции, в которых проецирование осуществляется на криволинейные поверхности или же проецирование осуществляется не с помощью прямых (такие проекции используются, например, в картографии — рисунки 54, 55[32]).

[32] Следует отметить, что, приводя иллюстрации к данной главе, используются те же самые проекции, методы построения которых описаны. Сделано это для того, чтобы материал выглядел более ясным, чем при отсутствии рисунков как таковых.

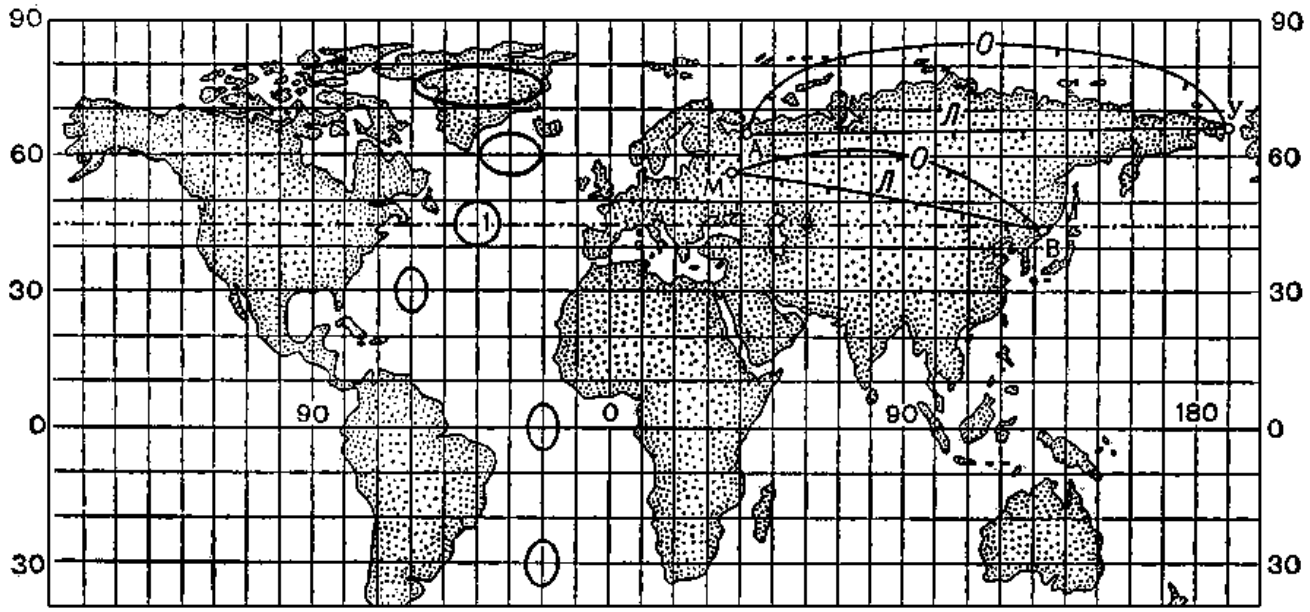


Рисунок 54 — Цилиндрическая равнопромежуточная (прямоугольная) проекция.

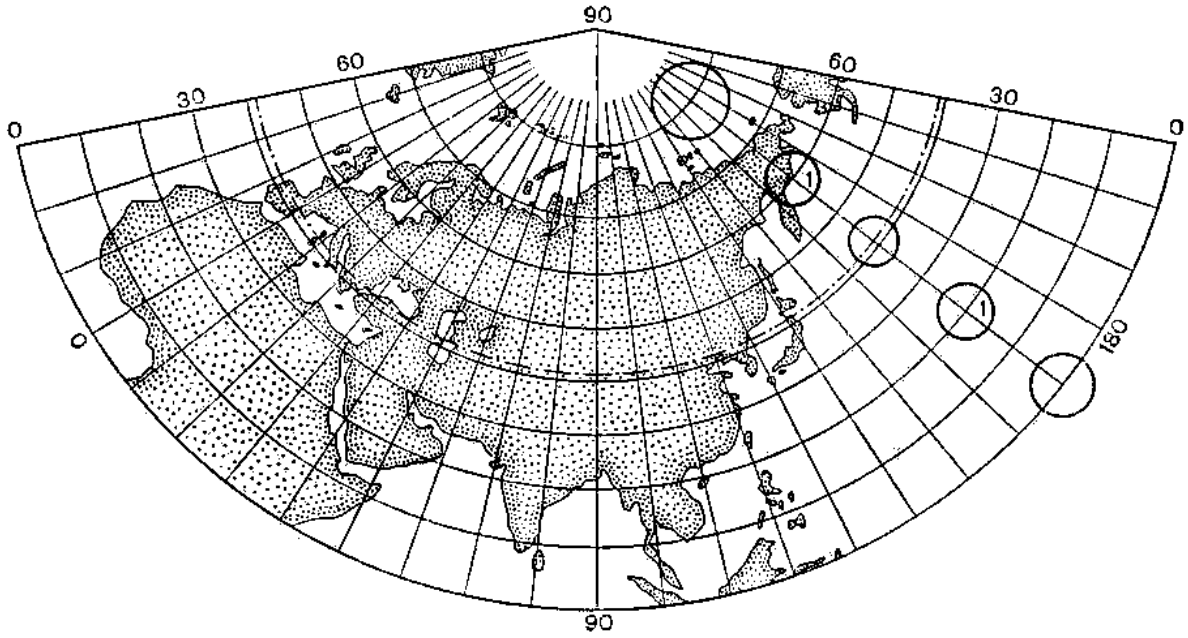


Рисунок 55 — Коническая равноугольная проекция.

Плоские геометрические проекции подразделяются на два основных класса: центральные и параллельные. Различие между ними определяется соотношением между центром проекции и проекционной плоскостью. Если расстояние между ними конечно, то проекция будет центральной, если же оно бесконечно, то проекция будет параллельной. Параллельные проекции названы так потому, что центр проекции бесконечно удален и все проекторы параллельны. При описании центральной проекции явно задаётся центр проекции, в то время как, определяя параллельную проекцию, указывается направление проецирования. На рисунках 56 и 57 показаны две различные проекции одного и того же отрезка, а также

проекторы, проходящие через его конечные точки. Поскольку проекция отрезка сама является отрезком, то достаточно спроецировать одни лишь конечные точки и соединить их.

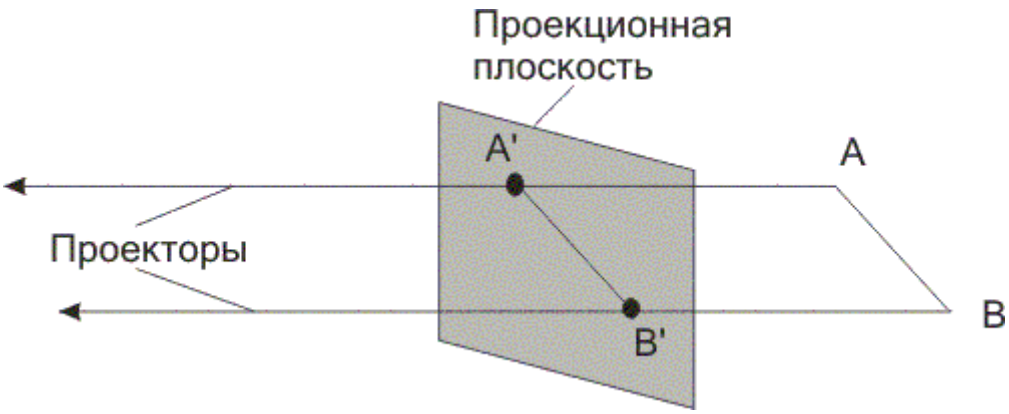


Рисунок 56 — Параллельная проекция.

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

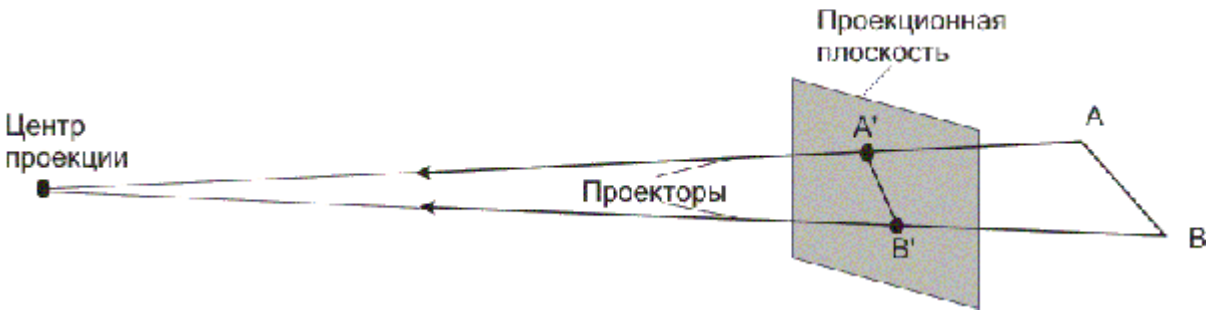


Рисунок 57 — Центральная проекция.

Параллельная проекция порождает менее реалистичное изображение, поскольку отсутствует перспективное укорачивание, хотя при этом могут иметь место различные постоянные укорачивания вдоль каждой из осей. Проекция фиксирует истинные размеры (с точностью до скалярного множителя), и

параллельные прямые остаются параллельными. Как и в случае центральной проекции, углы сохраняются только на тех гранях объекта, которые параллельны проекционной плоскости.

1.1 Параллельные проекции

Параллельные проекции разделяются на два типа в зависимости от соотношения между направлением проецирования и нормалью к проекционной плоскости. Если эти направления совпадают, т.е. направление проецирования является нормалью к проекционной плоскости, то проекция называется ортографической[33]. Если же проекторы не ортогональны к проекционной плоскости, то проекция называется косоугольной.

В инженерной графике наиболее широко используемыми видами ортографических проекций являются вид спереди, вид сверху (план) и вид сбоку, в которых проекционная плоскость перпендикулярна главным координатным осям, совпадающим вследствие этого с направлением проецирования (рисунок 58). Поскольку каждая проекция отображает лишь одну сторону объекта, часто совсем непросто представить себе пространственную структуру проецируемого объекта, даже если рассматривать сразу несколько проекций одного и того же объекта. Но, тем не менее, такие чертежи позволяют определять реальные размеры объекта.

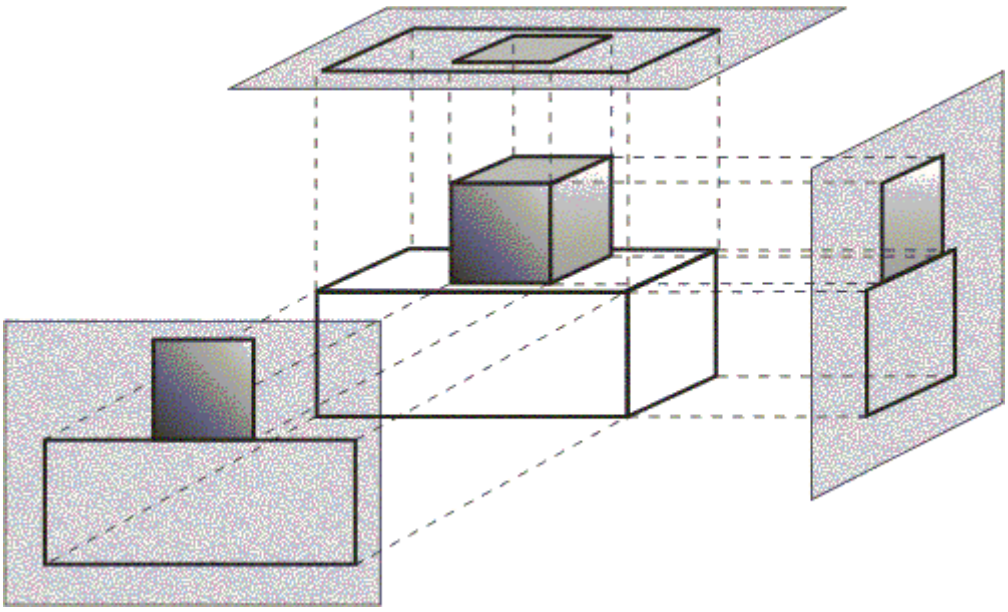


Рисунок 58 — Ортографические проекции.

[33] Орто- — греч. ορθός — прямой.

В случае аксонометрических [34] ортографических проекций используются проекционные плоскости, неперпендикулярные главным координатным осям, поэтому на них изображается сразу несколько сторон объекта, так же как и при центральном проецировании, однако в аксонометрии укорачивание постоянно, тогда как в случае центральной проекции оно связано с расстоянием от центра проекции. При аксонометрическом проецировании сохраняется параллельность прямых, а углы изменяются; расстояния же можно измерить вдоль каждой из главных координатных осей (в общем случае с различными масштабными коэффициентами).

Аксонометрические проекции подразделяются на три группы в соответствии с расположением проекционной плоскости по отношению к осям координат. Если нормаль к проекционной плоскости образует три различных угла с осями, то проекция называется триметрической (триметрией). Если два из этих углов одинаковы, то получаем диметрическую проекцию (диметрию). И, наконец, если все три угла равны между собой, то проекция называется изометрической (изометрией [35]). Изометрическая проекция обладает тем свойством, что все три главные координатные оси одинаково укорачиваются. Поэтому можно проводить измерения вдоль направления осей с одним и тем же масштабом. Кроме того, главные координатные оси проецируются так, что их проекции составляют равные углы друг с другом (рисунок 59).

[34] Аксон- — греч. ἄξων — ось.

[35] Изометрия — греч. ισος — равный, μετρέω — измеряю.

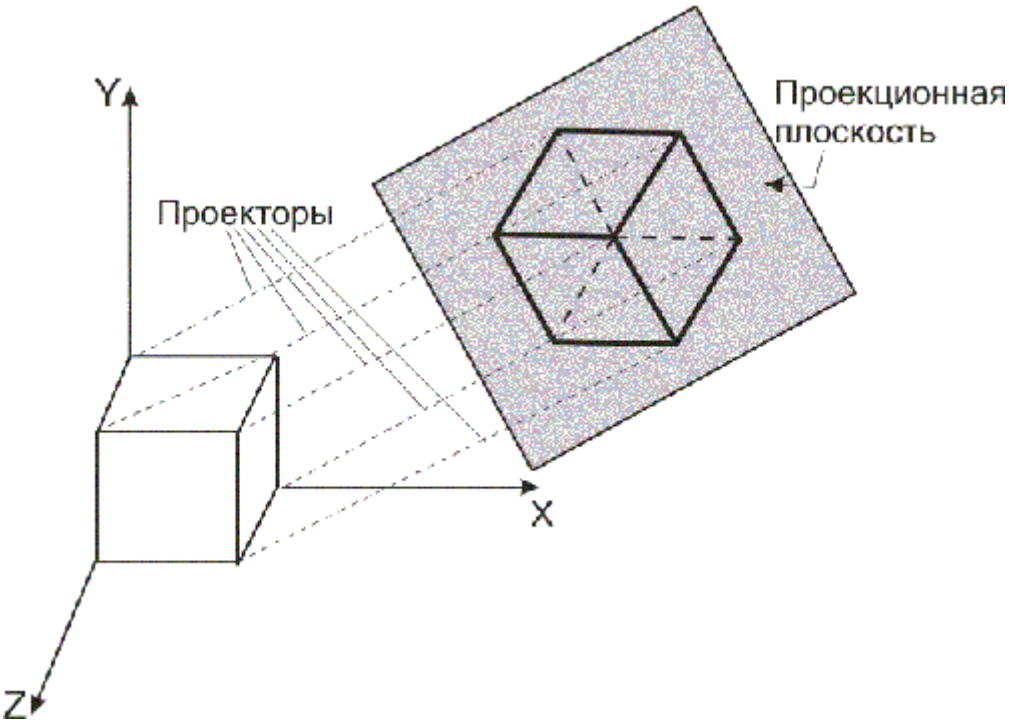


Рисунок 59 — Изометрическая проекция.

Косоугольные проекции также являются параллельными, причём проекционная плоскость перпендикулярна главной координатной оси. Сторона объекта, параллельная этой плоскости, проецируется так, что можно измерять углы и расстояния. Проецирование других сторон объекта также допускает проведение линейных измерений (но не угловых) вдоль главных осей.

Рассмотрим наиболее часто используемые косоугольные проекции: проекции кавалье (cavalier[36]) и кабине (cabinet[37]). В отечественной практике эти проекции называют горизонтальной косоугольной изометрией и кабинетной проекцией.

В проекции горизонтальной косоугольной изометрии направление проецирования составляет с плоскостью угол 45°. В результате проекция отрезка, перпендикулярного проекционной плоскости, имеет ту же длину, что и сам отрезок, т.е. укорачивание отсутствует (рисунок 60a). Кабинетная проекция имеет направление проецирования, которое составляет с проекционной плоскостью угол arctg(1/2). При этом отрезки, перпендикулярные проекционной плоскости, после проецирования составляют 1/2 их действительной длины, что более соответствует нашему визуальному опыту, поэтому изображение выглядит более реалистично (рисунок 60b).

[36] Фр. — косоугольный, аксонометрический; с высокой точки зрения, с птичьего полета (о перспективе, виде, проекции).

[37] Фр. — кабинет, рабочая комната.

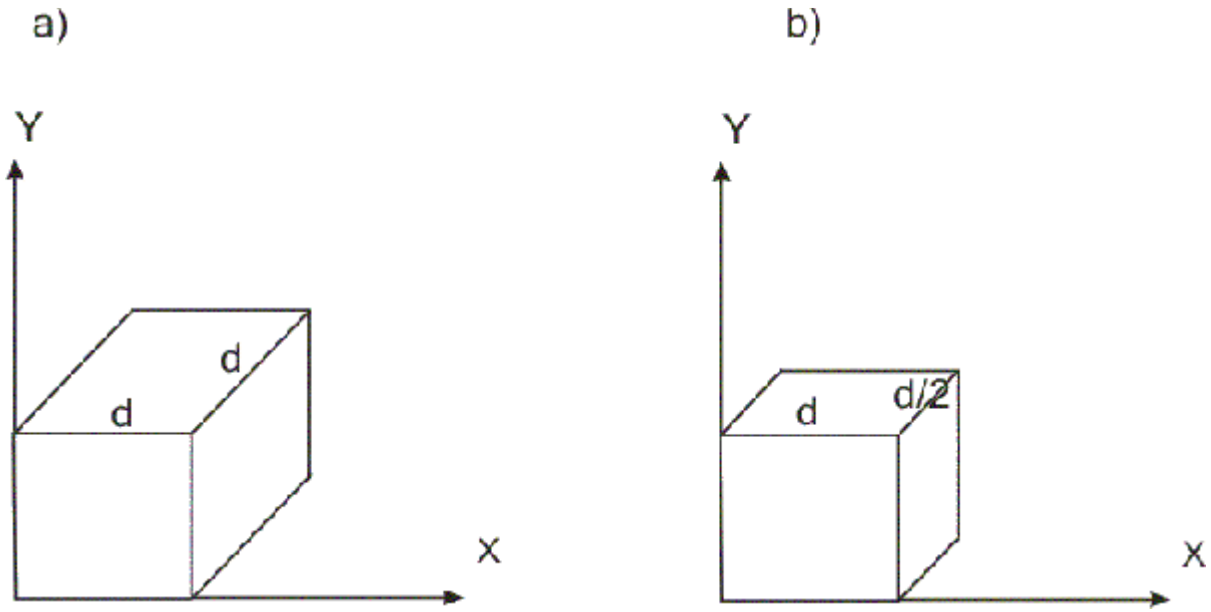


Рисунок 60 — Горизонтальная косоугольная изометрия (a) и кабинетная проекция (b).

1.2 Центральные проекции

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

Центральные проекции классифицируются в зависимости от числа главных точек схода, которыми они обладают, а, следовательно, и от числа координатных осей, которые пересекает проекционная плоскость. На рисунке 61 приведены три различные одноточечные проекции куба, причём две из них имеют одну точку схода, а третья — две точки.

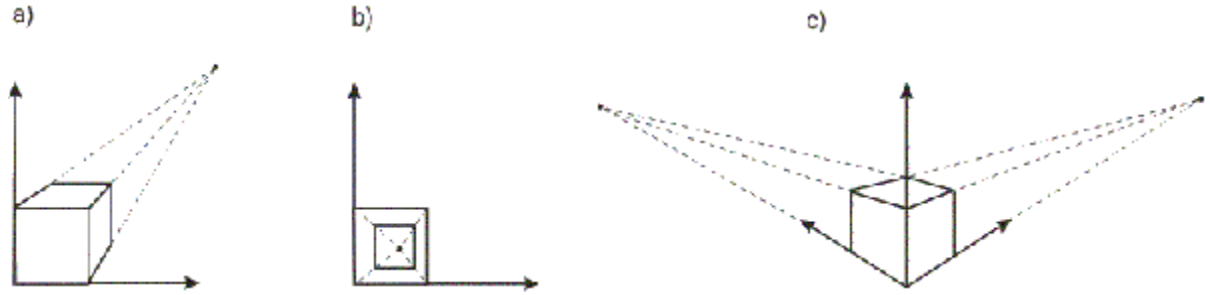


Рисунок 61 — Одноточечные и двухточечная проекции.

Двухточечная центральная проекция широко применяется в архитектурном, инженерном и промышленном проектировании и в рекламных изображениях, в которых вертикальные прямые проецируются как параллельные и, следовательно, не сходятся. Трёхточечные центральные проекции (рисунок 62) почти совсем не используются, во-первых, потому, что их трудно конструировать, а во-вторых, из-за того, что они добавляют мало нового с точки зрения реалистичности по сравнению с двухточечной проекцией.

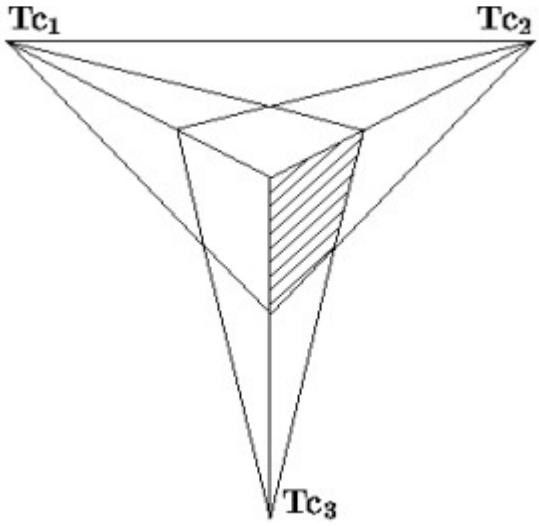


Рисунок 62 — Трёхточечная центральная проекция.

На рисунке 63 в виде общей схемы представлена классификация плоских проекций, используемая в компьютерной графике.

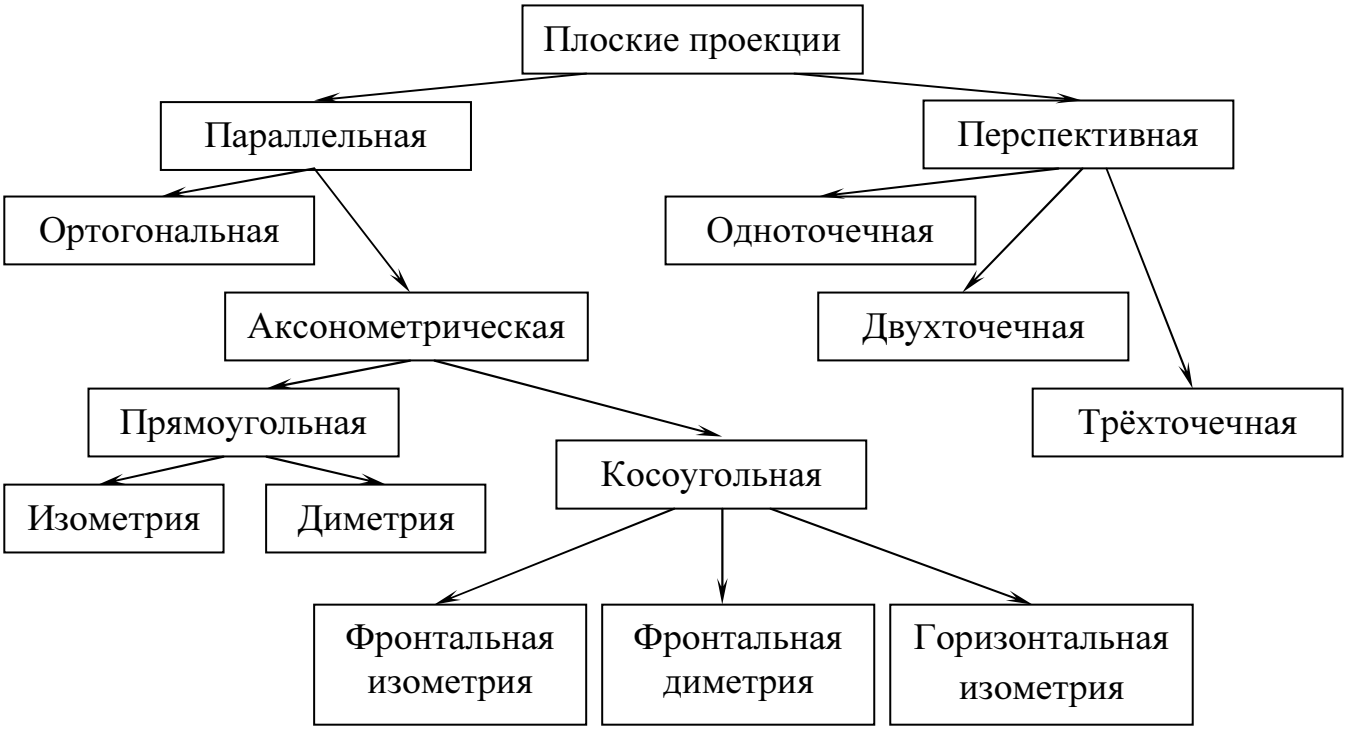


Рисунок 63 — Классификация плоских проекций.

2 Математический аппарат

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

2.1 Ортогональные проекции

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

Вектор, направленный по нормали от начала координат до пересечения с плоскостью, есть $\vec{N} = d \cdot \vec{n} = \left(n_x d, n_y d, n_z d\right) = (N_x, N_y, N_z)$.

Координаты вектора единичной нормали являются её направляющими косинусами.

Проецирование в пространстве однородных координат осуществляется следующей последовательностью шагов:

- Сдвиг на вектор $-\vec{N}$ с помощью матрицы S_1:

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

- Поворот, совмещающий направление нормали с направлением оси *OZ*. Этот поворот можно реализовать в виде двух поворотов: первый (относительно оси *OZ*) переводит нормаль в плоскость *YOZ*, а затем — поворот относительно оси *OY* до совмещения нормали с осью *OZ*. Соответствующую матрицу вращения, являющуюся произведением двух матриц, обозначим *R*.
- Проекция на плоскость *XOY* с помощью матрицы P_{xy}:

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

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

Полное преобразование, таким образом, определяется матрицей *Pr*:

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

2.2 Косоугольные проекции

Рассмотрим косоугольную проекцию на плоскость *XOY*, при которой орт[38] $\vec{e}_3 = (0,0,1)$ переходит в вектор $\vec{r}_0 = (a, b, 0)$, т.е. направление проекции задается вектором $\vec{p} = \vec{r}_0 - \vec{e}_3 = (a, b, -1)$. Такое преобразование в пространстве однородных координат можно задать с помощью матрицы *P*:

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

[38] Орт — единичный вектор, направленный вдоль соответствующей оси.

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

Глава 8. Растровое преобразование графических примитивов

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

1 Алгоритм Брезенхема растровой дискретизации отрезка

При построении растрового образа отрезка необходимо, прежде всего, установить критерии "хорошей" аппроксимации. Первое требование состоит в том, что отрезок должен начинаться и кончаться в заданных точках и при этом выглядеть сплошным и прямым (при достаточно высоком разрешении дисплея этого можно добиться). Кроме того, яркость вдоль отрезка должна быть одинаковой и не зависеть от наклона отрезка и его длины. Это требование выполнить сложнее, поскольку горизонтальные и вертикальные отрезки всегда будут ярче наклонных, а постоянная яркость вдоль отрезка опять же достигается на вертикальных, горизонтальных и наклоненных под углом в 45° линиях (рисунок 64). И, наконец, алгоритм должен работать быстро. Для этого необходимо по возможности исключить операции с вещественными числами. С целью ускорения работы алгоритма можно также реализовать его на аппаратном уровне.

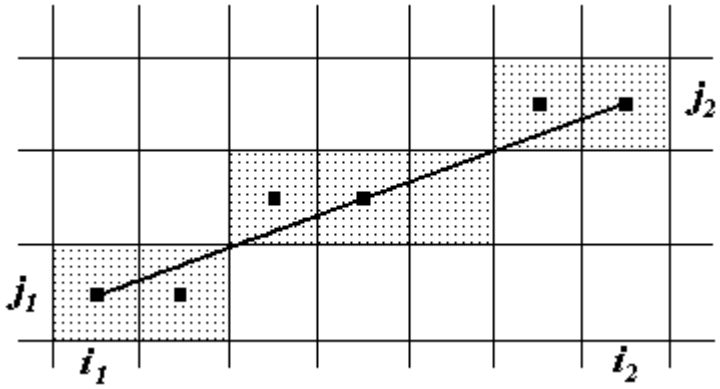


Рисунок 64 — Растровый образ отрезка.

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

Задача состоит в построении отрезка, соединяющего на экране точки с координатами $(i_1, j_1),(i_2, j_2)$ (будем считать, что $i_1 \neq i_2$). Для построения отрезка

прямой на плоскости с вещественными координатами можно воспользоваться уравнением прямой, проходящей через две заданные точки, которое имеет вид:

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

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

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

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

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

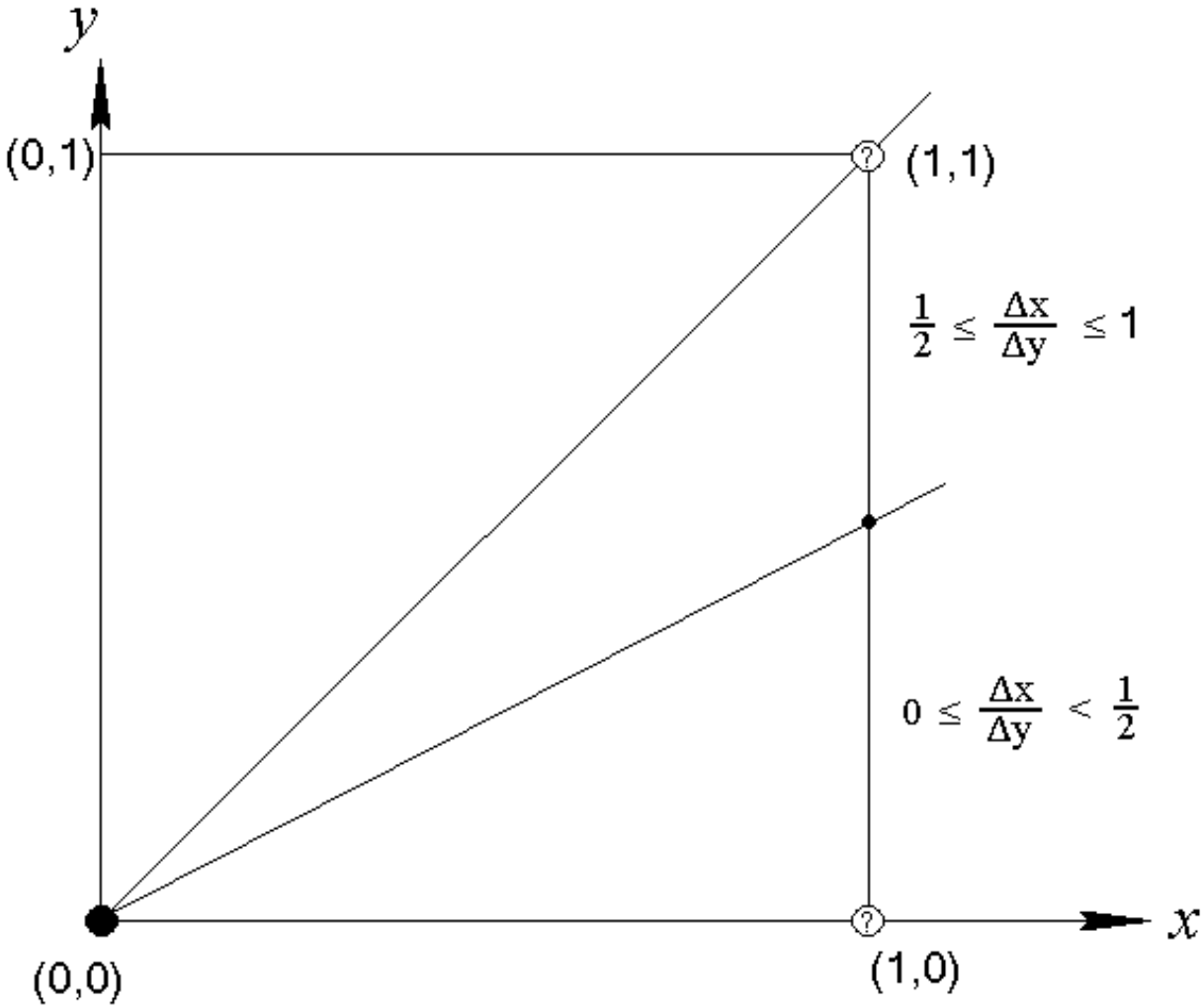


Рисунок 65 — Связь углового коэффициента с выбором пикселя.

На рисунке 65 это иллюстрируется для отрезка с угловым коэффициентом, лежащим в диапазоне от 0 до 1. Из рисунка можно заметить, что если угловой коэффициент $k \geq 1/2$, то при выходе из точки (0, 0) пересечение с прямой $x = 1$ будет ближе к прямой $y = 1$, чем к прямой $y = 0$. Следовательно, точка растра (1, 1) лучше аппроксимирует прохождение отрезка, чем точка (1, 0). При $k < 1/2$ верно обратное.

На рисунке 66 показано, каким образом строятся точки растра для отрезка с тангенсом угла наклона 3/8, а на рисунке 67 — график смещения. В начале построения смещение полагается равным $k - 1/2$, а затем на каждом шаге оно наращивается на величину k, и если при этом вертикальная координата точки растра увеличивается на единицу, то смещение в свою очередь уменьшается на единицу.

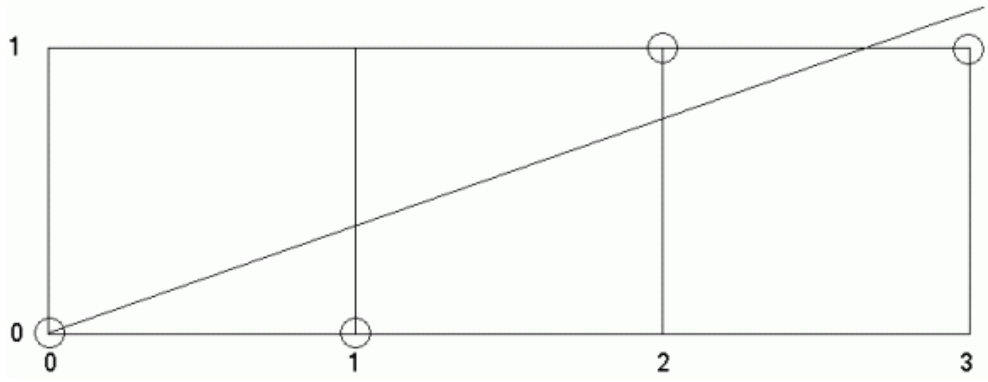


Рисунок 66 — Пиксели, принадлежащие развертке отрезка.

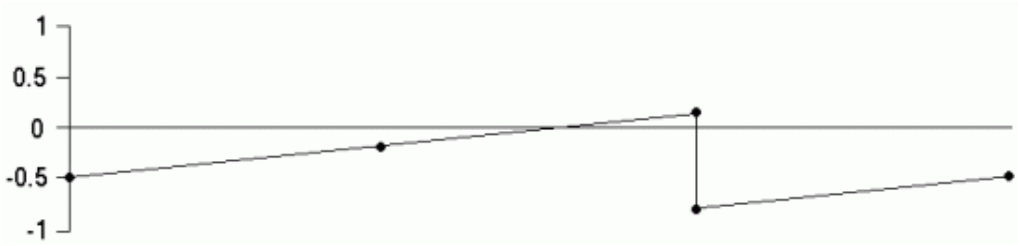


Рисунок 67 — График изменения отклонения.

На рисунке 68 приведена блок-схема алгоритма для случая $i_2 > i_1$, $j_2 > j_1$, $k > 0$. Нетрудно понять, как от этого алгоритма перейти к целочисленному: достаточно вместо величины смещения e перейти к величине $\tilde{e} = 2\Delta i \cdot e$.

Ниже приведён общий алгоритм Брезенхема, который учитывает все возможные случаи направления отрезка, рассматриваемого как вектор на координатной плоскости (на рисунке 69 выделены четыре области и указаны особенности алгоритма в каждой из них).

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

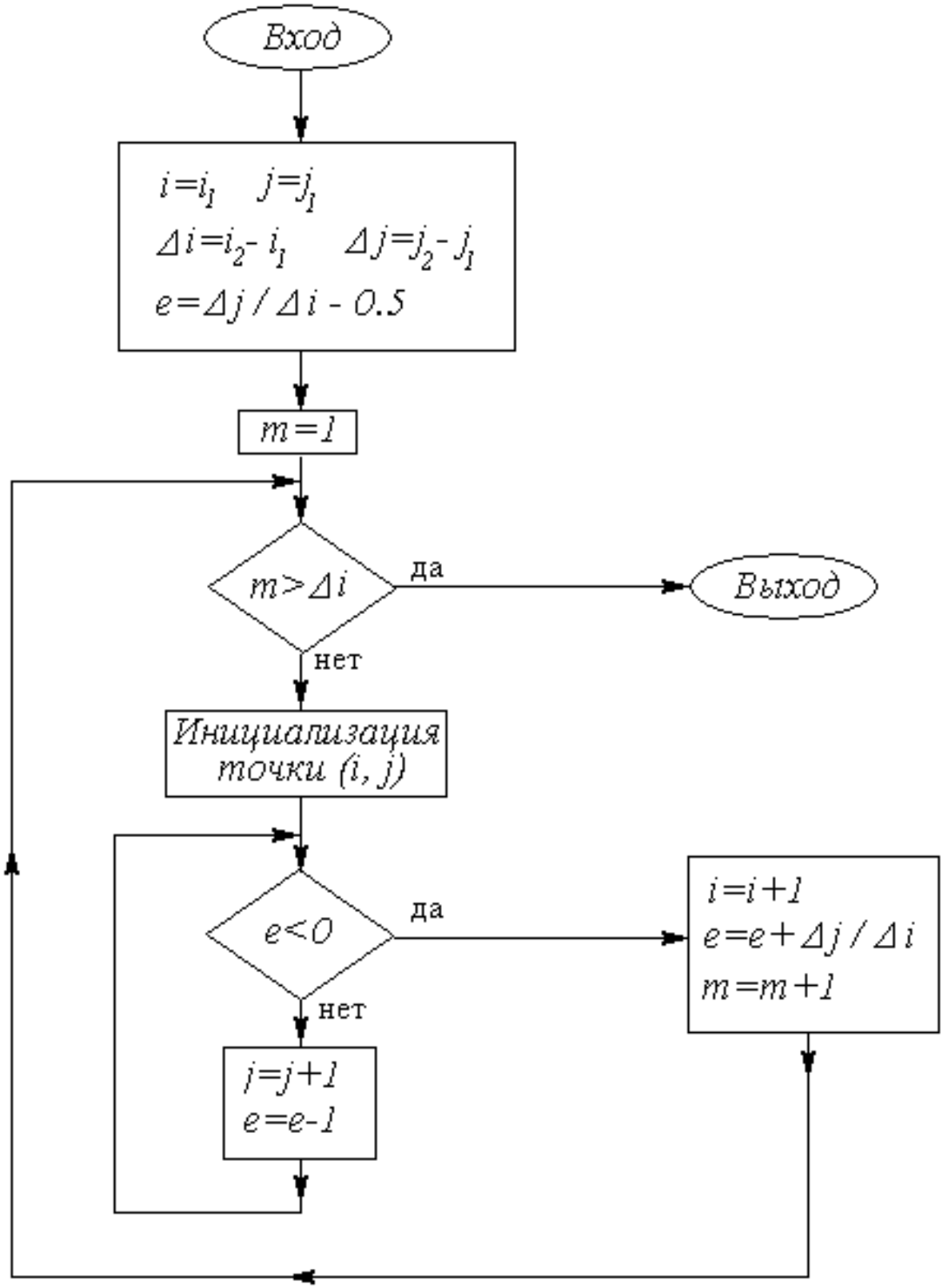


Рисунок 68 — Блок-схема одной ветви алгоритма Брезенхема.

В описании алгоритма используются следующие функции:

`swap(a, b)` — обмен значений переменных *a*, *b*;

`abs(a)` — абсолютное значение *a*;

`sign(a)` — 0, если $a = 0$, 1, если $a > 0$, –1, если $a < 0$;

`point(i, j)` — инициализация точки (i, j).

Общий алгоритм Брезенхема:

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

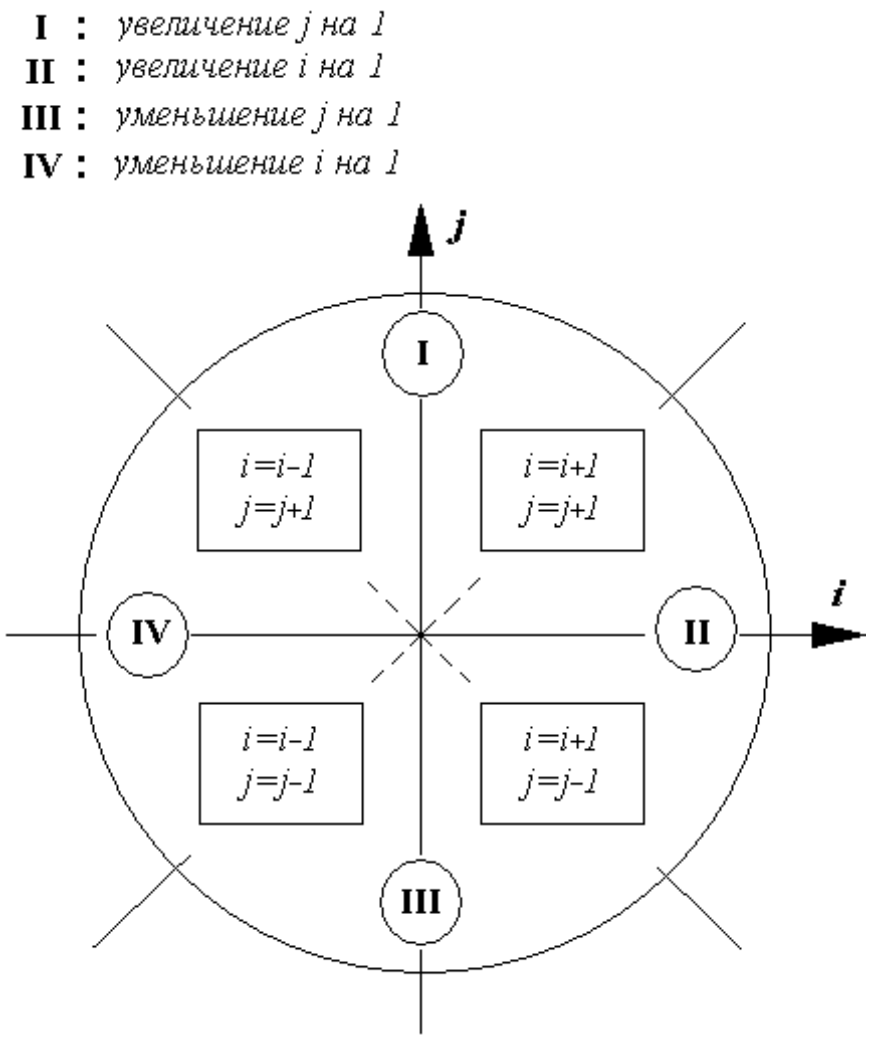


Рисунок 69 — Четыре возможных направления отрезка.

2 Алгоритмы Брезенхема растровой дискретизации окружности и эллипса

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

При выбранном направлении движения по окружности имеется только три возможности для расположения ближайшего пикселя: на единицу вправо, на единицу вниз и по диагонали вниз (рисунок 70).

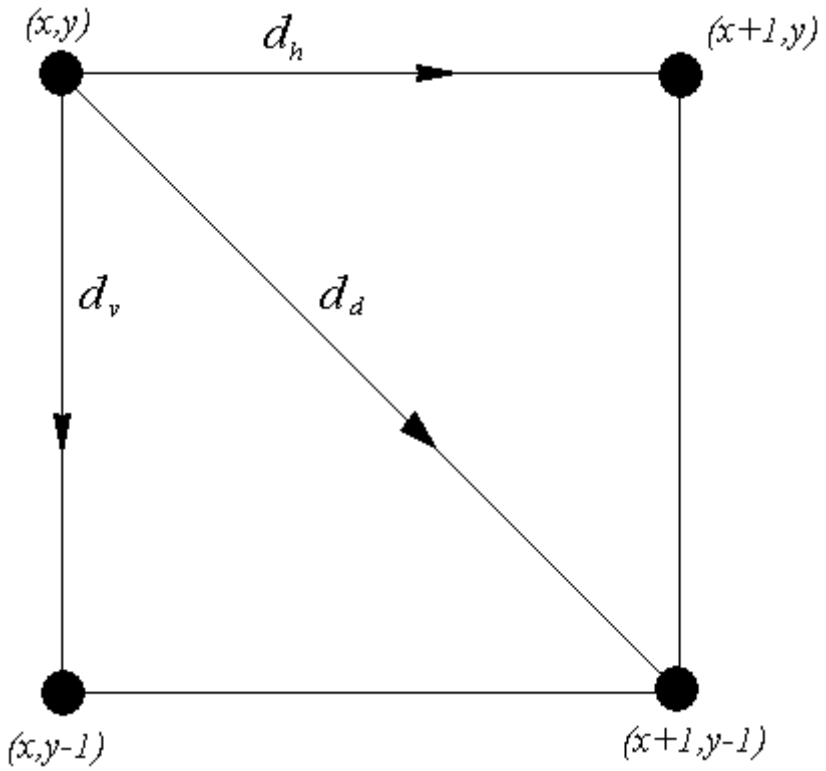


Рисунок 70 — Выбор ближайшего пикселя при движении по окружности.

Выбор варианта можно осуществить, вычислив расстояния до этих точек и выбрав минимальное из них:

$$d_h = |s_h|,\ d_v = |s_v|,\ d_d = |s_d|,$$

$$s_h = (x+1)^2 + y^2 - r^2,\ s_v = x^2 + (y-1)^2 - r^2,\ s_d = (x+1)^2 + (y-1)^2 - r^2.$$

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

горизонтальном переходе $(x' = x + 1, y' = y)$ — $s'_d = s_d + 2x' + 1$, при вертикальном $(x' = x, y' = y - 1)$ — $s'_d = s_d - 2y' + 1$.

Все оставшиеся части окружности строятся параллельно: после получения очередной точки (x', y') можно инициализировать ещё семь точек с координатами $(-x', y')$, $(-x', -y')$, $(x', -y')$, (y', x'), $(y', -x')$, $(-y', -x')$, $(-y', x')$.

Блок-схема данного алгоритма приведена на рисунке 71.

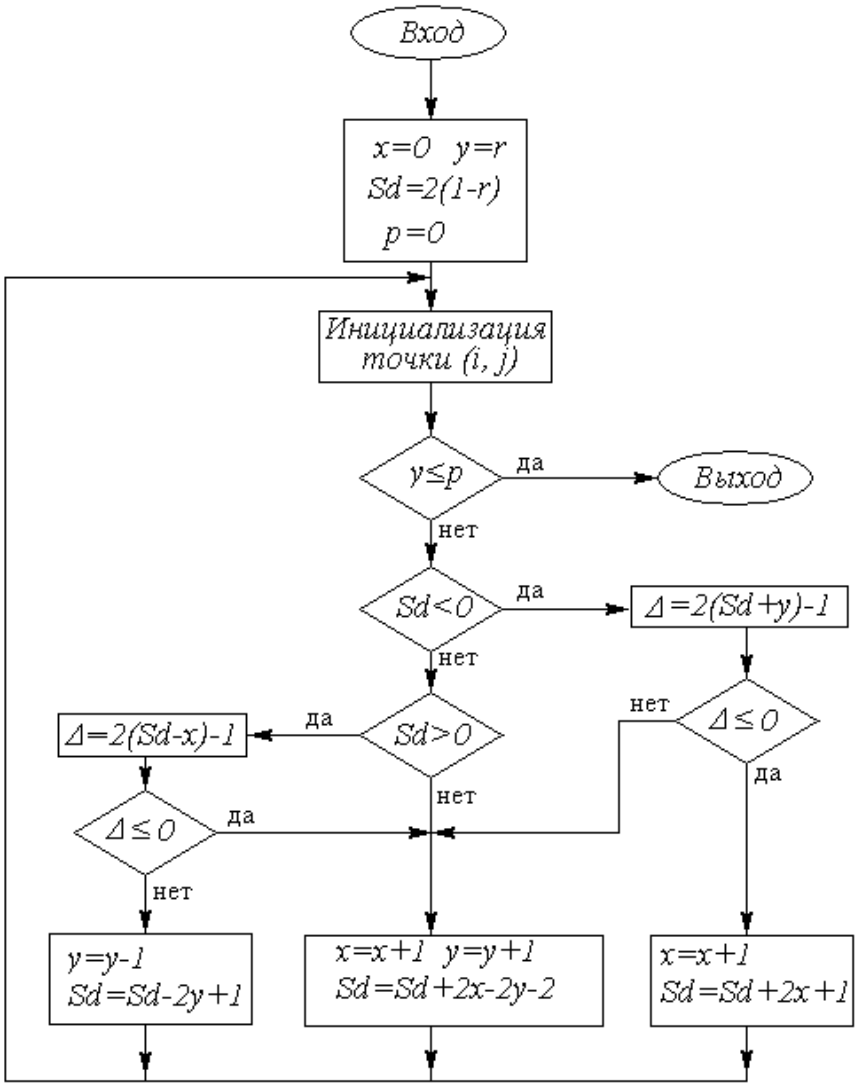


Рисунок 71 — Блок-схема построения восьмой части окружности.

Пример работы алгоритма Брезенхема генерации окружности

На рисунке П.5 приведён пример работы пошагового алгоритма Брезенхема генерации окружности с радиусом равным 8 и центром в начале координат для первого квадранта.

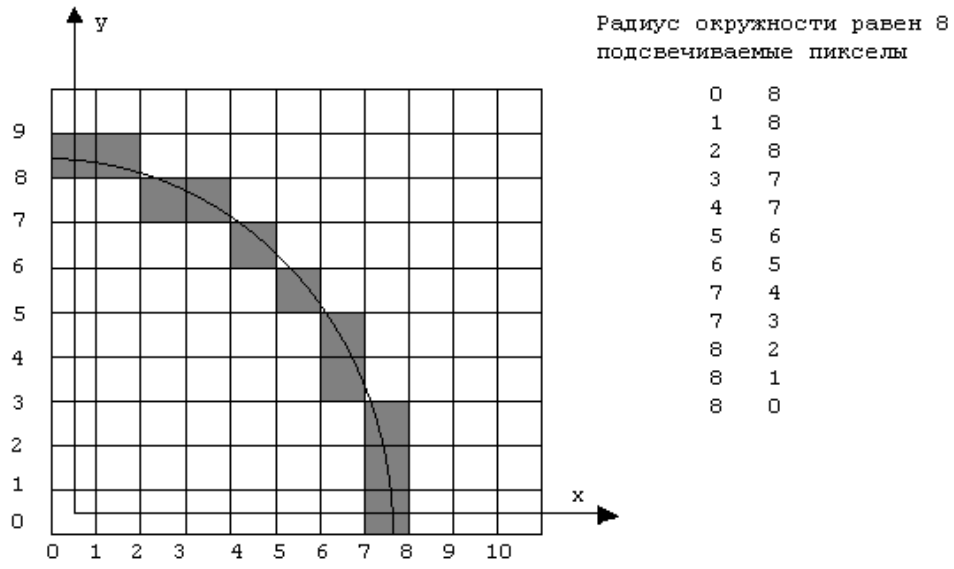


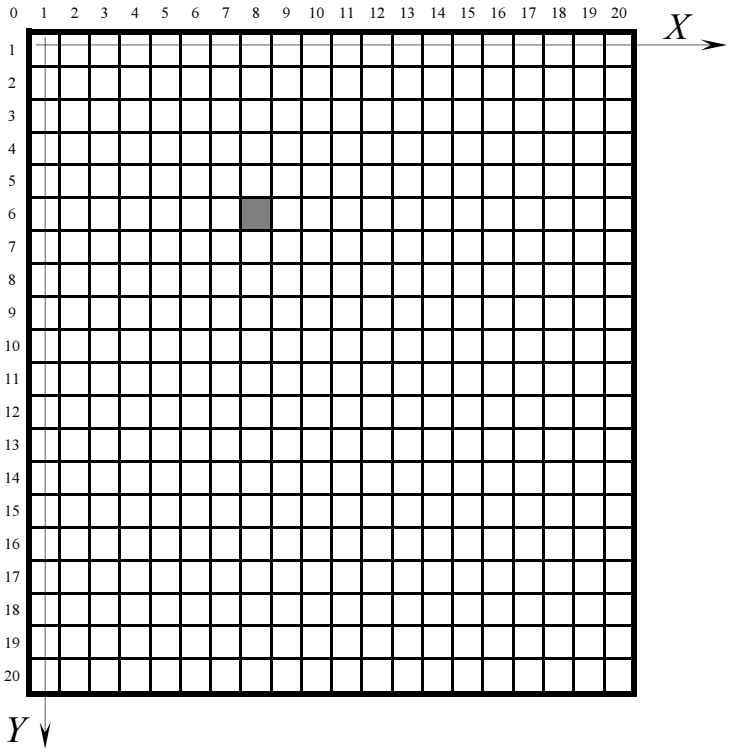
Рисунок П.5 — Результаты работы пошагового алгоритма Брезенхема генерации окружности для первого квадранта.

Задача:

Даны: координаты центра окружности $R = (8; 6)$, радиус окружности $r = 9.7$, экран с разрешением 20 x 20.

Используя пошаговый алгоритм Брезенхема отобразить на экране окружность.

Вид экрана с отмеченным центром окружности:



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

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

которое перепишем в виде:

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

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

В каждой точке (x, y) эллипса существует вектор нормали, задаваемый градиентом функции f. Дугу разобьём на две части: первая — с углом между нормалью и горизонтальной осью больше 45° (тангенс больше 1) и вторая — с углом, меньшим 45° (рисунок 72). Движение вдоль дуги будем осуществлять в направлении по часовой стрелке, начиная с точки (0, b). Вдоль всей дуги координата является монотонно убывающей функцией от x, но в первой части она убывает медленнее, чем растет аргумент, а во второй — быстрее. Поэтому при построении растрового образа в первой части будем увеличивать x на единицу и искать соответствующее значение y, а во второй — сначала уменьшать значение y на единицу и определять соответствующее значение x.

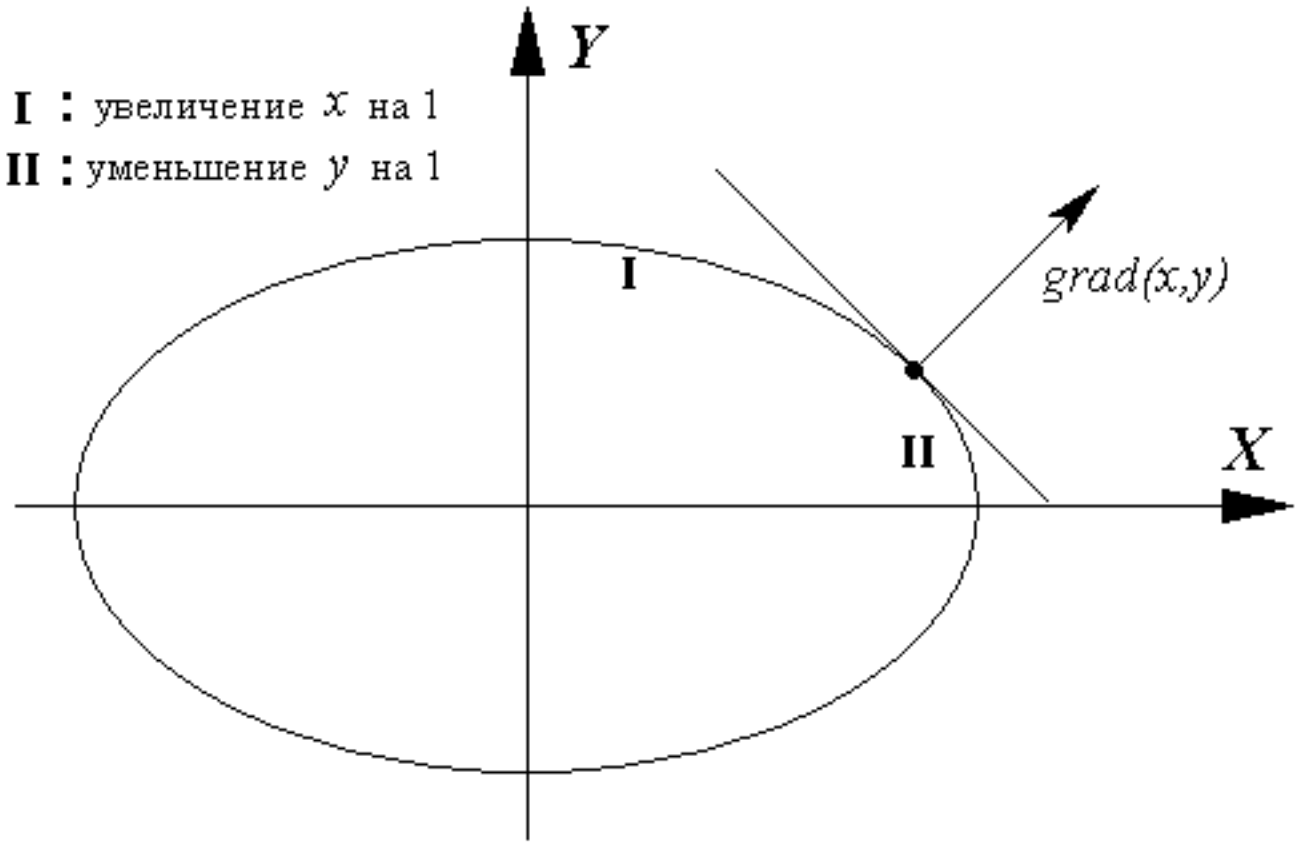


Рисунок 72 — Две области на участке эллипса.

Направление нормали соответствует вектору[39]:

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

Отсюда находим тангенс угла наклона вектора нормали:

$$t = \frac{a^2y}{b^2x}.$$

Приравнивая его единице, получаем, что координаты точки деления дуги на вышеуказанные части удовлетворяют равенству:

$$b^2x = a^2y.$$

Поэтому критерием того, что необходимо перейти ко второй области в целочисленных координатах, будет соотношение:

$$a^2(y - 1/2) \le b^2(x + 1),$$

или, переходя к целочисленным операциям, $a^2(2y - 1) \le 2b^2(x + 1)$.

При перемещении вдоль первого участка дуги необходимо из каждой точки перейти либо по горизонтали, либо по диагонали, и критерий такого перехода напоминает тот, который использовался при построении растрового образа окружности. Находясь в точке (x, y), нужно вычислить значение $\Delta = f(x + 1, y - 1/2)$. Если это значение меньше нуля, то дополнительная точка $(x+1, y - 1/2)$ лежит внутри эллипса, следовательно, ближайшая точка растра есть $(x + 1, y)$, в противном случае это точка $(x + 1, y - 1)$ (рисунок 73а).

На втором участке дуги возможен переход либо по диагонали, либо по вертикали, поэтому, сначала значение координаты y уменьшается на единицу, затем вычисляется $\Delta = f(x + 1/2, y - 1)$ и направление перехода выбирается аналогично предыдущему случаю (рисунок 73б).

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

---

[39] Градиент (от лат. gradiens, род. падеж gradientis — шагающий) — характеристика, показывающая направление наискорейшего возрастания некоторой величины, значение которой меняется от одной точки пространства к другой. Например, если взять высоту поверхности Земли над уровнем моря (2-мерное пространство), то её градиент в каждой точке поверхности будет показывать "в горку". Как видно из объяснения, градиент является векторной функцией, а величина, которую он характеризует — функцией скалярной.

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

Для второй половины дуги получим:

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

Все оставшиеся дуги эллипса строятся параллельно: после получения очередной точки (x', y'), можно инициализировать ещё три точки с координатами $(-x', y')$, $(-x', -y')$, $(x', -y')$.

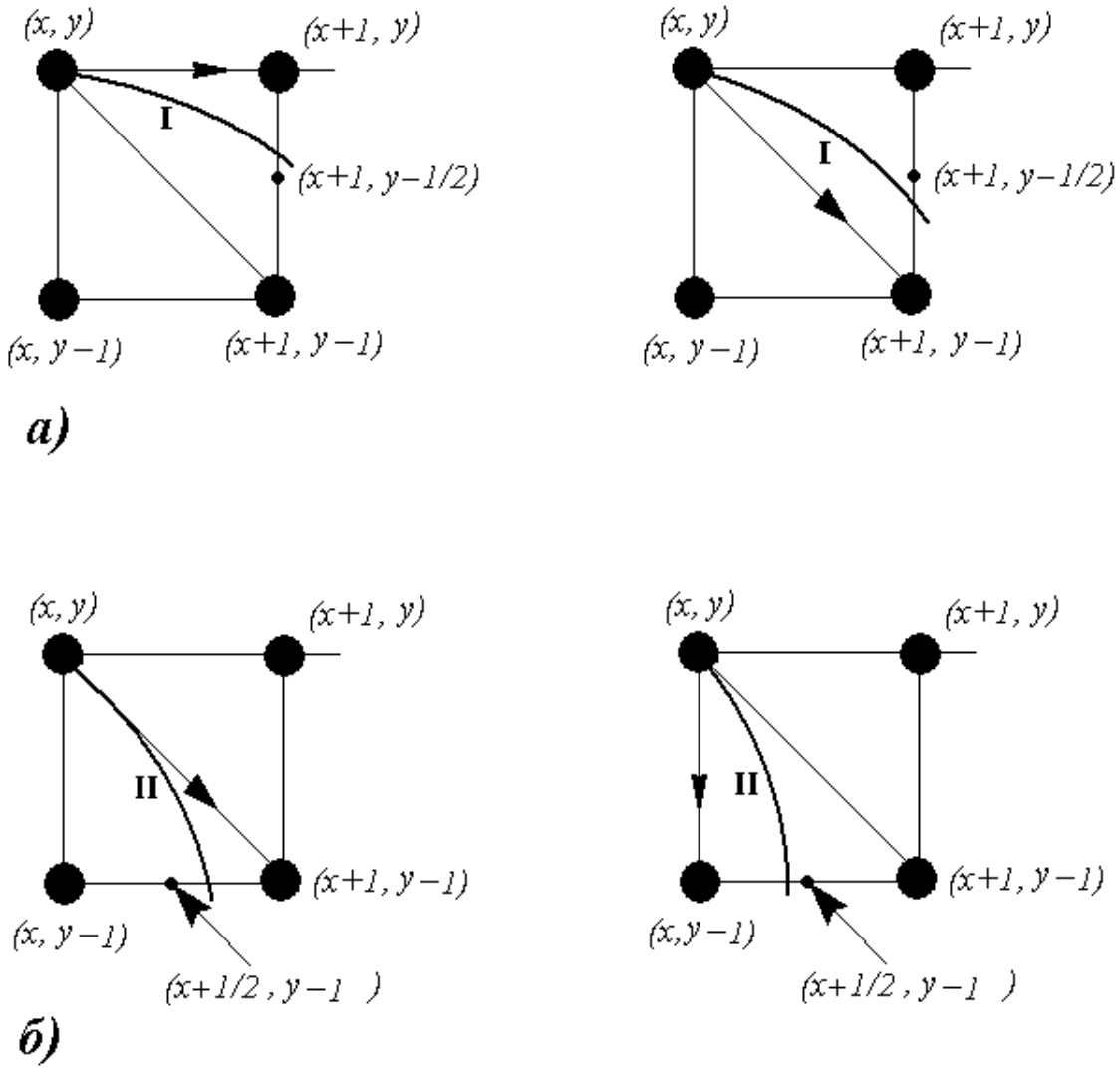


Рисунок 73 — Схема перехода в первой и второй областях дуги эллипса.

3 Алгоритмы заполнения областей

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

3.1 Метод затравки

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

По способу задания области метод делятся на два типа:

- гранично-определенные, задаваемые своей (замкнутой) границей такой, что коды пикселей границы отличны от кодов внутренней, перекрашиваемой части области. На коды пиксели внутренней части области налагаются два условия — они должны быть отличны от кода пикселей границы и кода пикселя перекраски. Если внутри гранично-определенной области имеется ещё одна граница, нарисованная пикселями с тем же кодом, что и внешняя граница, то соответствующая часть области не должна перекрашиваться;
- внутренне-определенные, нарисованные одним определенным кодом пикселя. При заполнении этот код заменяется на новый код закраски.

Заполняемая область или её граница — некоторое связное множество пикселей. По способам доступа к соседним пикселям области делятся на 4-х и 8-ми связные.

В 4-х связных областях доступ к соседним пикселям осуществляется в четырех направлениях — горизонтально влево и вправо и вертикально вверх и вниз. В 8-ми связных областях к этим направлениям добавляются ещё 4 диагональных. Используя тот или иной тип связности, двигаясь от точки затравки, закрашиваются все пиксели области.

Важно отметить, что для 4-х связной прямоугольной области граница 8-ми связна (рисунок 74а) и, наоборот, у 8-ми связной области граница 4-х связна (рисунок 74б). Поэтому заполнение 4-х связной области 8-ми связным алгоритмом

может привести к "просачиванию" через границу и закраске пикселей в примыкающей области.

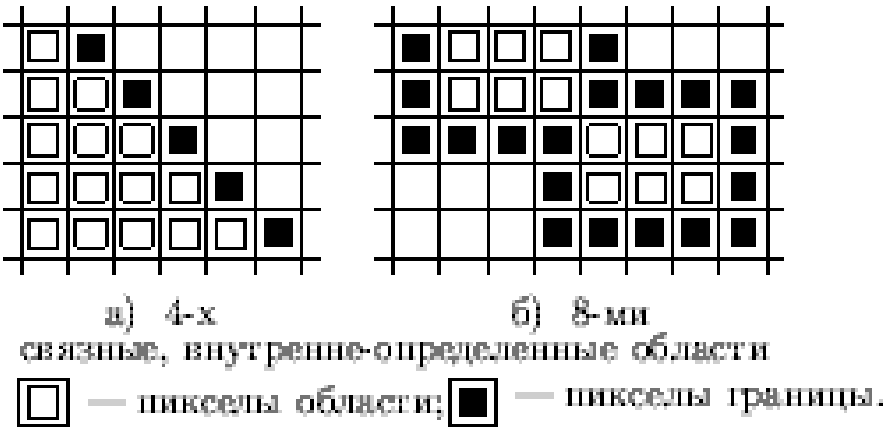


Рисунок 74 — Связность областей и их границ.

Следует знать, что 4-х связную область можно заполнить как 4-х, так и 8-ми связным алгоритмом. Обратное же неверно. Так область на рисунке 74а можно заполнить любым алгоритмом, а область на рисунке 74б, состоящую из двух примыкающих 4-х связных областей можно заполнить только 8-ми связным алгоритмом.

Рассмотрим простой алгоритм заполнения с затравкой с использованием стека. Под стеком в данном случае понимаем массив, в который можно последовательно помещать значения и последовательно извлекать, причём извлекаются элементы не в порядке поступления, а наоборот: по принципу "первым пришёл – последним ушёл" ("first in – last out").

В алгоритме используется стек. В него заносится затравочный пиксель. Логика работы алгоритма закраски 4-х связной гранично-определенной области следующая:

1. Начало алгоритма заполнения с затравкой.
2. Поместить координаты затравочного пикселя в стек.
3. Начало цикла заполнения области:
   Пока стек не пуст:
   3.1 Извлечь координаты пикселя из стека.
   3.2 Перекрасить пиксель.
4. Для всех четырех соседних пикселей проверить: является ли он граничным или уже перекрашен. Если не перекрашен, то занести его координаты в стек.
5. Конец цикла заполнения.
6. Конец алгоритма заполнения.

Недостатками алгоритма с затравкой являются его "медленность" и расход большого количества памяти для стека, а также в стеке запоминаются пиксели дублирующие друг друга.

3.2 Метод растровой развёртки

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

Рассмотрим методы растровой развёртки в применении к заполнению многоугольников. Простейший метод построения состоит в том, чтобы для каждого пикселя растра проверить его принадлежность внутренности многоугольника. Но такой перебор слишком неэкономичен, поскольку фигура может занимать лишь незначительную часть экрана, а геометрический поиск — задача трудоемкая, сопряженная с длинными вычислениями. Алгоритм станет более эффективным, если предварительно выявить минимальный прямоугольник, в который погружен контур многоугольника, но и этого может оказаться недостаточно.

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

В качестве примера приведём алгоритм закраски внутренней области треугольника, основанный на составлении полного упорядоченного списка всех отрезков, составляющих этот треугольник. Для записи горизонтальных координат концов этих отрезков будем использовать два массива X_{min} и X_{max} размерностью, равной числу пикселей растра по вертикали (рисунок 75).

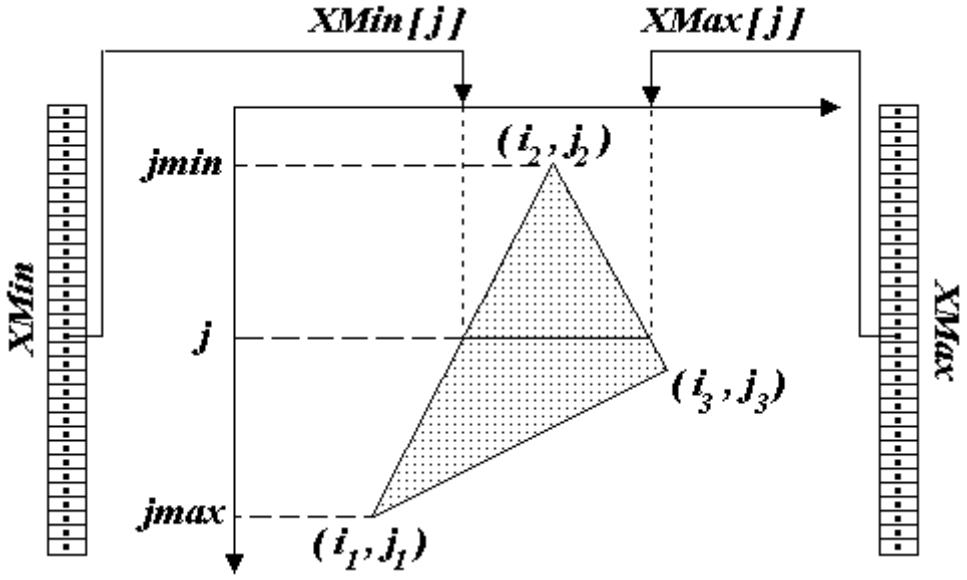


Рисунок 75 — Схема построения растровой развертки треугольника.

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

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

Этот алгоритм можно легко распространить на случай произвольного выпуклого многоугольника.

Преимущество этого алгоритма в том, что каждый пиксель обрабатывается строго один раз.

Раздел 5. Алгоритмы закраски геометрических объектов. Работа со светом

Глава 9. Закрашивание. Рендеринг полигональных моделей

Визуальное восприятие объектов окружающей действительности представляет собой сложный процесс, имеющий как физические, так и психологические аспекты.

Глаз адаптируется к средней яркости рассматриваемой сцены, поэтому при смене фона изменяется восприятие сцены. Например, однородно окрашенная область на более тёмном фоне будет казаться более яркой, чем на светлом. Кроме того, она будет восприниматься как более обширная (рисунок 76).

Рисунок 76 — Эффекты восприятия изображения.

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

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

1 Простая модель освещения

Объекты окружающего пространства становятся видимыми для глаза благодаря световой энергии, которая может излучаться поверхностью предмета, отражаться или проходить сквозь неё. В свою очередь, отражение света от поверхности зависит от физических свойств материала, из которого она изготовлена, а также от характера и расположения источника света. Яркость (или

интенсивность) освещения зависит от энергии светового потока, которая обуславливается, во-первых, мощностью источника света, а во-вторых, отражающими и пропускающими свойствами объекта.

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

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

Свет точечного источника отражается от поверхности рассеивателя по закону Ламберта: интенсивность отражения пропорциональна косинусу угла между внешней нормалью к поверхности и направлением к источнику света (рисунок 77).

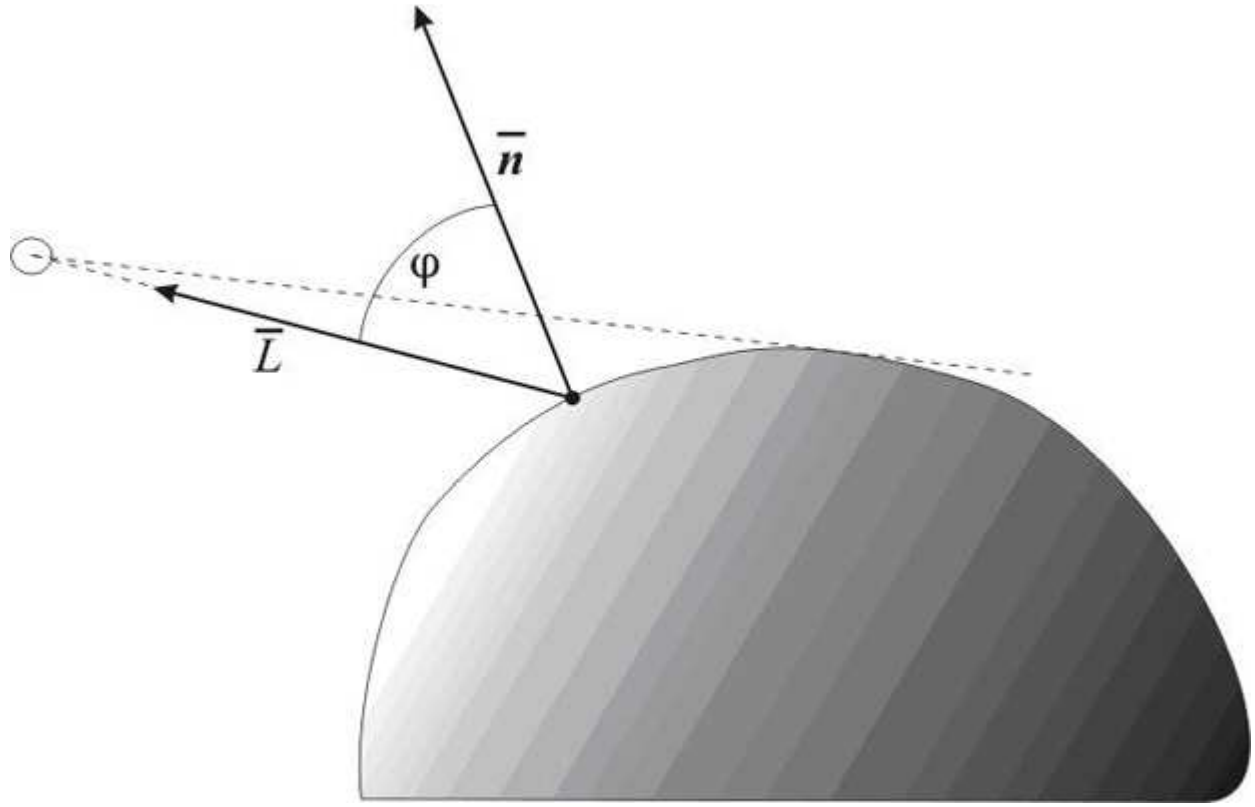


Рисунок 77 — Освещение точечным источником.

Если I_S — интенсивность источника света, φ — угол между вектором внешней нормали к поверхности и направлением к источнику света, то интенсивность отраженного света определяется формулой (33):

$$I = \begin{cases} I_S \cos(\phi) \text{ при } 0 \le \phi \le \frac{\pi}{2} \\ 0 \text{ — в противном случае} \end{cases}. \quad (33)$$

При таком расчёте интенсивности получится очень контрастная картина, т.к. участки поверхности, на которые лучи от источника не попадают напрямую, останутся абсолютно черными. Для повышения реалистичности необходимо учитывать рассеивание света в окружающем пространстве. Поэтому вводится фоновая освещенность, зависящая от интенсивности рассеянного света I_F, и интенсивность отраженного света определяется выражением (34):

$$I = \begin{cases} I_F k_F + k_S I_S \cos(\phi) \text{ при } 0 \le \phi \le \frac{\pi}{2} \\ I_F k_F \text{ — в противном случае} \end{cases}, \quad (34)$$

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

В описанной модели пока никак не учитывалась удалённость источника света от поверхности, поэтому по освещённости двух объектов нельзя судить об их взаимном расположении в пространстве. Если нужно получить перспективное изображение, то необходимо включить затухание интенсивности с расстоянием. Обычно интенсивность света обратно пропорциональна квадрату расстояния от источника. В качестве расстояния до источника в случае перспективного преобразования можно взять расстояние до центра проекции, и если он достаточно удалён, то изображение будет достаточно адекватным. Но если этот центр расположен близко к объекту, то квадрат расстояния меняется очень быстро, и в этом случае лучше использовать линейное затухание. Тогда интенсивность отражённого света от непосредственно освещённых участков поверхности будет задаваться формулой (35):

$$I = I_F k_F + \frac{k_S I_S \cos(\phi)}{d + C}, \quad (35)$$

где d — расстояние от центра проекции;
C — произвольная постоянная.

Если центр проекции находится на бесконечности, т.е. при параллельном проецировании, то в качестве d можно взять расстояние до объекта, наиболее близкого к наблюдателю.

В отличие от диффузного, зеркальное отражение является направленным. Идеальное зеркало отражает лучи по принципу "отраженный и падающий лучи лежат в одной плоскости, причём угол падения равен углу отражения" (имеется в виду угол между направлением луча и нормалью к поверхности). Если поверхность не идеально зеркальная, то лучи отражаются в различных направлениях, но с разной интенсивностью, а функция изменения интенсивности имеет четко выраженный максимум. Поскольку физические свойства зеркального отражения довольно сложны, то в компьютерной графике используется эмпирическая модель Фонга. Суть её заключается в том, что для глаза наблюдателя интенсивность зеркально отраженного луча зависит от угла между идеально отраженным лучом и направлением к наблюдателю (рисунок 78а).

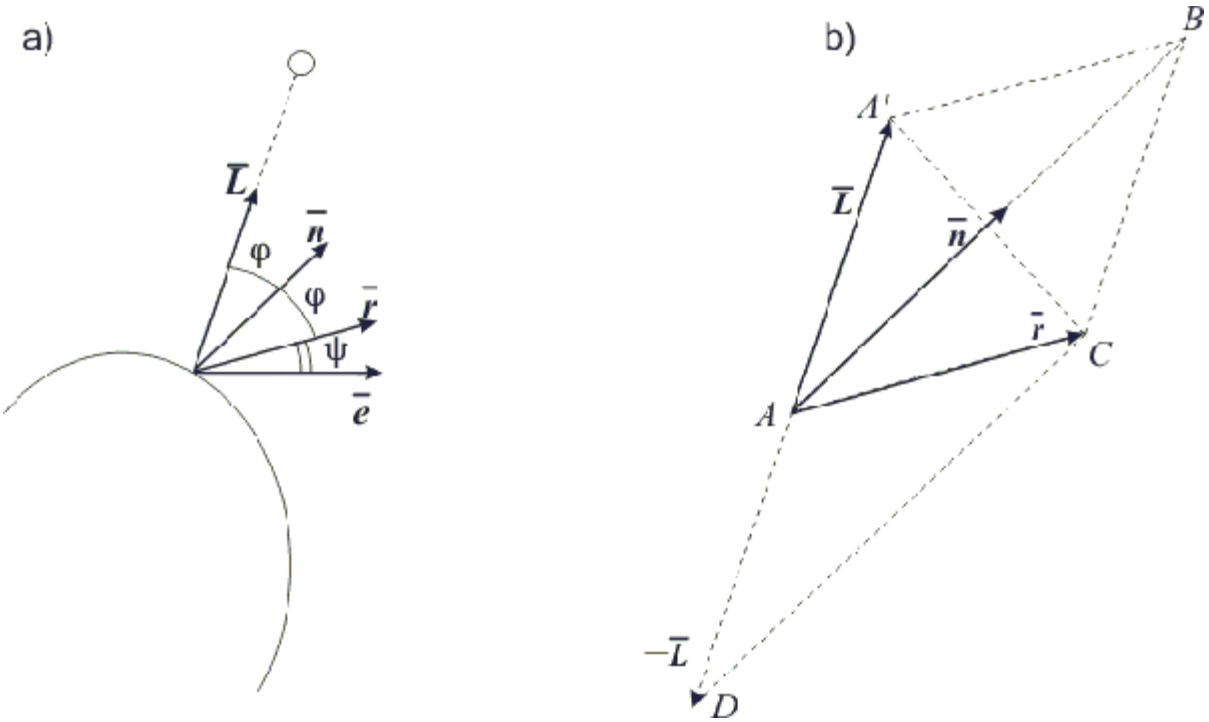


Рисунок 78 — Зеркальное отражение.

Кроме того, поскольку зеркальное отражение зависит ещё и от длины волны, это также будем учитывать в формуле для вычисления интенсивности. Модель Фонга описывается соотношением (36):

$$I_Z = \omega(\varphi, \lambda)\, I_S \cos^n(\psi) \qquad (36)$$

где $\omega(\varphi, \lambda)$ — функция отражения;

λ — длина волны;

n — степень, влияющая на размеры светового блика, наблюдаемого зрителем.

Графики этой функции приведены на рисунке 79, и они как раз являются характерными кривыми поведения функции изменения интенсивности в зависимости от свойств поверхности.

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

$$I = I_F k_F + \frac{I_S(k_s \cos(\phi) + \omega(\phi,\lambda)\cos(\psi))}{d+C}, \quad (37)$$

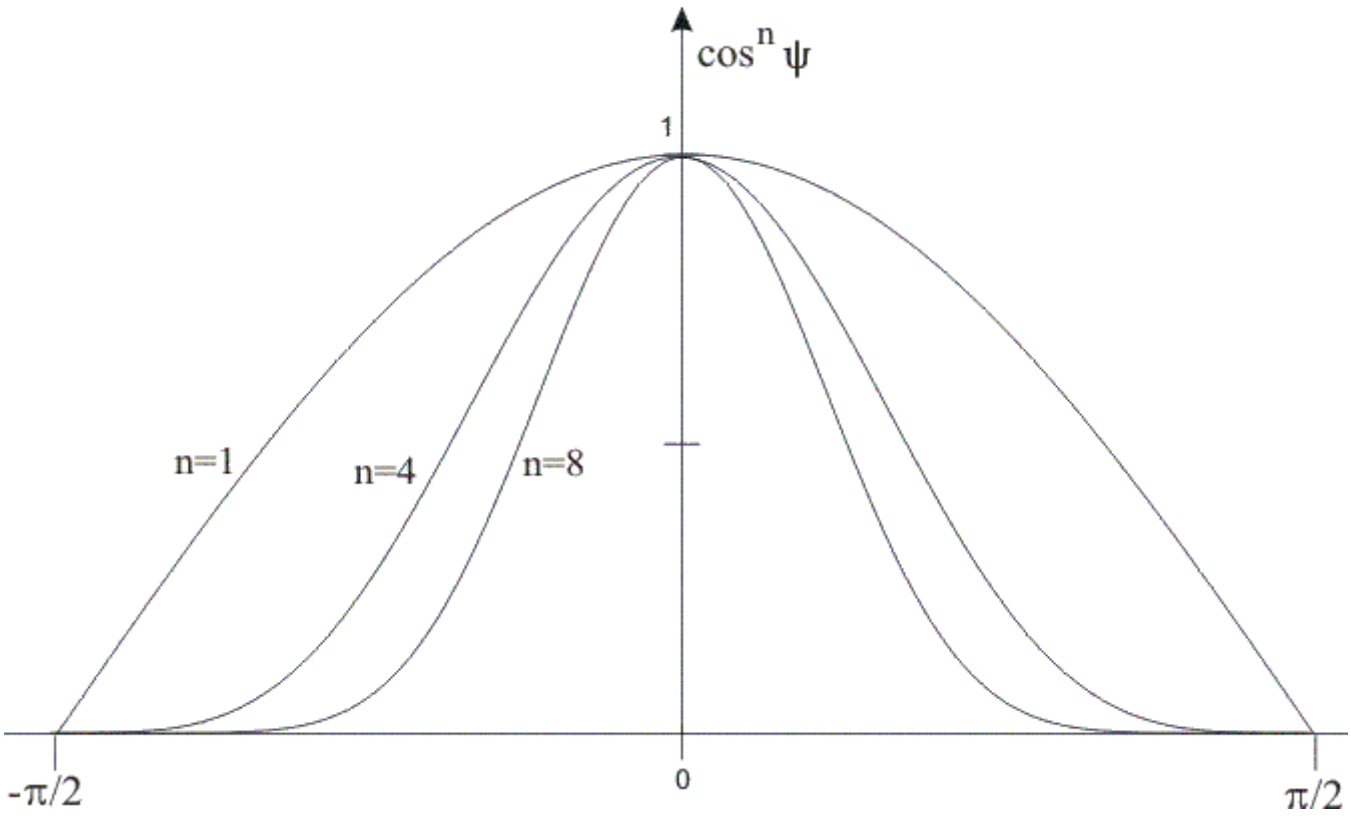


Рисунок 79 — График зависимости модели Фонга для зеркального отражения.

Используя единичные векторы $\vec{L}$ (направление к источнику) и $\vec{n}$ (внешняя нормаль), косинус угла φ можно вычислить через скалярное произведение: $\cos(\phi) = (\overrightarrow{L} \cdot \vec{n})$.

Для расчета интенсивности зеркального отражения сначала надо определить отраженный вектор $\vec{r}$. Из рисунка 78b видно, что $\vec{r} = |AB| \cdot \vec{n} - \vec{L}$. С другой стороны AB является диагональю ромба $AA'BC$, поэтому $|AB| = 2(\overrightarrow{L} \cdot \vec{n})$. Учитывая все эти соотношения, получаем формулу:

$$I = I_F k_F + \frac{I_S(k_s\vec{L}\cdot\vec{n}+\omega(\phi,\lambda)\left(2(\vec{L}\cdot\vec{n})\cdot(\vec{e}\cdot\vec{n})-(\vec{e}\cdot\vec{L})\right)^{\mathrm{n}})}{d+C}. \quad (38)$$

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

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Основная литература

1. Красильников Н.Н., Цифровая обработка 2D- и 3D-изображений: учебное пособие. – Спб.: БХВ-Петербург, 2011, 608с.

2. Роджерс Д., Алгоритмические основы машинной графики. М.: Мир, 1989, 512 с.

3. Шикин Е.В., Боресков А.В. Компьютерная графика. – М.: Диалог-МИФИ, 1996. 288 с.

4. Абламейко С.В., Лагуновский Д.М. Обработка изображений: технология, методы, применение. Мн.: Амалфея, 2000, 304 с.

Дополнительная литература

1. Bresenham, J.E. A Linear Algorithm for Incremental Digital Display of Circular Arcs / J.E. Bresenham // Communication of the ACM. – 1977. – V. 20(2). – P. 100–106.

2. Danielson P.E. Incremental curve generation / P.E. Danielson // IEEE Transaction on Computers. – 1970. – V. 19. – P. 23–28.

3. Suenaga, Yasuhito. A High Speed Algorithm for the Circular Arcs / Yasuhito Suenaga, Takahiko Kamae, Tomonori Kabayashi // IEEE Trans. on Computers. – 1979. – V. 28(10). – P. 728–736.

4. Pitteway, M.L.V. Integer circles etc. – Three move of Breseriham's algorithm / M.L.V. Pitteway, R.J. Botting // Computer Graphics and Image Processing. – 1974. – V. 3. – P. 260–261.

5. Horn, B.K.P. Circle generators for display devices / B.K.P. Horn // Computer Graphics, and Image Processing. – 1976. – V. 5. – P. 580–588.

6. Badler, N.I. Disk generators for a raster display device / N.I. Badler // Computer, Graphics and Image Processing. – 1977. – V. 6. – P. 589–593.

7. Chung W.L. On circle generation algorithms / W.L. Chung // Computer Graphics and Image Processing. – 1977. – V. 6. – P. 196–198.

8. Doros, M. Algorithms tor Generation to Discrete Circles, Rings and Disks / M. Doros // Computer Graphics and Image Processing. – 1979. – V. 10(4). – P. 366–371.

9. Pitteway M. Bresenham's Algorithm with Gray Scale / M. Pitteway, D. Watkinson // Communication of the ACM. – 1980. – V. 23(11). – P. 625–626.

10. Moller G. Past digital vector and circle generator with binary rate multipliers / G. Moller // Computer Graphics. – 1978. – V.12(4). – P. 81–91.

11. McCrea P.G. On DDA circle generation for computer graphics / P.G. McCrea, P.W. Baker // IEEE Trans. on Computers. – 1975. – V. 24. – P. 1109–1110.

12. Maxwell, P.C. The Generation of Polygons Representing Circles. Ellipses and Hyperbolas / P.C. Maxwell,P.W. Baker // Computer Graphics end Image Processing. – 1979. – V. 10. – P. 84–93.

13. Foley, James D. Computer graphics – principles and practice / James D. Foley [etc] // ADDISON-WESLEY PUBLISHING COMPANY. 1990.

14. Божко, А.Н. Компьютерная графика : учеб. пособие для вузов / А.Н. Божко, Д.М. Жук, В.Б. Маничев. – М. : Изд-во МГТУ, 2007.

15. Фоли Дж. Основы интерактивной машинной графики / Дж. Фоли, А. ван Дам. В 2 т. – М. : Мир, 1985.

16. Шикин, Е.В. Компьютерная графика. Динамика, реалистические изображения / Е.В. Шикин, А.В. Боресков. – М. : ДИАЛОГ-МИФИ, 1995.

17. Шикин Е.В. Кривые и поверхности на экране компьютера: Рук. по сплайнам для пользователей / Е.В. Шикин, А.И. Плис. – М. : Диалог-МИФИ, 1996.

18. Шикин, Е.В. Начала компьютерной графики / Е.В. Шикин, А.В. Боресков, А.А. Зайцев. – М. : Диалог-МИФИ, 1993.

19. Чириков, С.В. Алгоритмы компьютерной графики (методы растрирования кривых) : учеб. пособие / С.В. Чириков. – СПб. : СПб ГИТМО(ТУ), 2001.

20. Джамбруно, М. Трехмерная графика и анимация : пер. с англ. / М. Джамбруно. – 2-е изд. – М. : Вильямс, 2002.

21. Никулин Е.А. Компьютерная геометрия и алгоритмы машинной графики / Е.А.Никулин. – СПб. : БХВ-Петербург, 2003.

22. Куликов А.И., Овчинникова Т.Э. Алгоритмические основы современной компьютерной графики / НОУ «ИНТУИТ», 2016. 231 с.