МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

И.А. Бессмертный

КОМПЬЮТЕРНАЯ ГРАФИКА

Учебное пособие

Санкт-Петербург

2011

И.А.Бессмертный. Компьютерная графика – СПб: СПбГУ ИТМО, 2010. – 88 с.

Настоящее учебное пособие разработано в рамках дисциплины «Компьютерная графика», преподаваемой на кафедре вычислительной техники СПбГУИТМО и включает в себя основные принципы отображения графики в вычислительной технике, основы программирования видеоадаптеров персональных компьютеров для отображения простейших изображений, а также принципы синтеза реалистичных изображений методами растеризации и трассировки лучей.

Для студентов специальностей 23010011, 23010013, 23010031, 23010032, 23010033, 023010035

Рекомендовано к печати ученым советом факультета КТиУ __.02.2011, протокол №_.

В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России, которым присвоена категория «Национальный исследовательский университет». Министерством образования и науки Российской Федерации была утверждена Программа развития **государственного образовательного учреждения высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» на 2009–2018 годы**.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Компьютерная графика это раздел вычислительной техники, посвященный проблемам преобразования данных в графическую форму, синтеза графических объектов, отображения и редактирования изображений. Известно, что с помощью зрения мы получаем около 90% всей информации об окружающем мире, и логично было потребовать, чтобы компьютер представлял данные в привычном для нас виде и понимал исходные данные, представленные в графической форме. Компьютер вторгается практически во все области человеческой деятельности, в т.ч. в такие, где без графического представления данных обойтись невозможно. Черный квадрат Малевича можно описать словами, и это поймут все. Функцию синуса можно отобразить в математической нотации, и поймут многие. Более сложные математические функции поймут только математики, но какими средствами описать географическую карту или фотопортрет?

Эволюция персональных компьютеров (ПК) привела к тому, что видеосистема стала сопоставимой по вычислительной мощности с центральным процессором, что позволяет выводить на экран фотореалистичные динамические объекты, синтезируемые на том же самом компьютере. Последним достижением компьютерной графики на момент написания данной работы является фильм Дж. Камерона «Аватар», почти полностью созданный с помощью компьютерной трехмерной анимации.

Целью данного учебного пособия является ознакомление читателей с основными принципами отображения графики, алгоритмами отображения графических объектов и принципами обработки изображений. Изучение принципов формирования изображений на самом низком уровне не только полезно для глубокого понимания работы продвинутых графических пакетов, например, Photoshop, но просто необходимо для программирования графических приложений для встроенных вычислительных систем, мобильных телефонов и других гаджетов. Данная цель оправдывает выбор в качестве объектов изучения таких режимов работы видеоадаптеров персональных компьютеров, которые уже давно не используются. Современные видеокарты ПК получают от программы не команду нарисовать точку, а отобразить полигон (многоугольник) с заданными параметрами заливки. Программировать графические приложения на таком уровне проще, но понимания работы видеоподсистемы этот не прибавляет.

Автор выражает глубокую благодарность Э.В. Стародубцеву на методическую помощь в подготовке этого издания.

СЛОВАРЬ ТЕРМИНОВ

ПИКСЕЛЬ – элемент изображения на экране (*Picture Cell* или *Picture Element*).

РАСТР – изображение, построенное из отдельных элементов (точек), как правило, расположенных регулярно. В большинстве приложений компьютерной графики, растровое изображение представляется двумерным массивом точек, цвет и яркость каждой из которых задаются независимо.

РЕНДЕРИНГ – финальный процесс в компьютерной графике, во время которого создается картинка в соответствии с 3D-сценой. Синоним слова «визуализация».

ЦВЕТОВАЯ МОДЕЛЬ – абстрактная модель описания представления цветов в виде кортежей чисел, обычно из трёх или четырёх значений, называемых цветовыми компонентами или цветовыми координатами. Вместе с методом интерпретации этих данных (например, определение условий воспроизведения и/или просмотра — то есть задание способа реализации), множество цветов цветовой модели определяет цветовое пространство.

ГЛАВА 1. ОСНОВНЫЕ ПРИНЦИПЫ ФОРМИРОВАНИЯ ИЗОБРАЖЕНИЙ

1.1. Что такое изоражение

Изображение – объект, образ, в той или иной степени подобный (но не идентичный) изображаемому объекту (ru.wikipedia.org). Данное определение отражает важнейшее свойство изображение – обобщение или абстрагирование от каких-либо свойств. В противном случае изображение будет просто клоном объекта. Степень абстрагирования может быть любой. Так, живописный портрет или фотография представляют собой абстрагирование от трехмерности, скульптура – от цвета, восковая фигура – от материала. Естественно, все изображения могут абстрагироваться от размера, хотя некоторые экскурсоводы на Дворцовой площади утверждают, что Александровскую колонну венчает фигура ангела в натуральную величину. Обобщение в первую очередь диктуется целью изображения объекта, а уже затем – техническими возможностями. Например, было бы странным, если бы вместо пиктограммы на двери туалета висел портрет конкретного человека, а манекены в магазине потели в жару.

Изображение является частным случаем **отражения** – философской категории, обозначающей способность материальных объектов изменяться в процессе взаимодействия. Отражение может быть непосредственным (след ноги на песке) или опосредованным (скульптура, как отражение объекта через ментальные и физические усилия скульптора). С математической точки зрения изображение – **отображение** или **функция,** воспроизводящая пространственную структуру объекта.

Пространственная структура объекта – взаимное расположение элементов объекта в пространстве. Очевидно, что объект должен существовать в природе или в воображении автора изображения. Последнее убедительно доказал Дж. Камерон в упомянутом фильме «Аватар». Пространственная структура объекта может быть описана через его элементы. Вспомним математическое определение прямой (объект) как геометрического места точек (элементов), одинаково расположенных относительно друг друга, или окружности, как геометрического места точек, равноудаленных от точки, называемой центром окружности, на величину радиуса. Пространственная структура объекта может быть иерархической: куб состоит из квадратов, квадраты из отрезков прямых, прямые из точек. Взаимное расположение может распространяться по иерархии. Например, положение ниппеля на велосипедном колесе может

быть определено по отношению к ступице этого колеса, но не может быть установлено относительно рамы.

Элементы объекта являются **смежными,** если образуют **связное пространство**, т.е. пространство, которое невозможно разбить на два непустых непересекающихся множества. Например, грани куба имеют общие ребра.

Между объектом A и его изображением B существуют следующие отношения:

1. Объект A состоит из множества элементов $\{a\}$, а объект B – из множества элементов $\{b\}$, причем каждому элементу b соответствует элемент a. Обратное высказывание ложно.
2. Каждой паре связных элементов из множества $\{b\}$ соответствует пара элементов из множества $\{a\}$. Обратное утверждение также неверно.

Данные отношения позволяют устанавливать и воспроизводить на изображениях существенные свойства объектов, например, голова у человека является смежной с шеей, причем это можно видеть и на скульптурах, и на фотографиях, и на пиктограммах олимпийских видов спорта. Правда, нас может поставить в тупик проблема изображения Солнечной системы: Луна не является смежной с Землей, однако, мы не имеем права изображать ее в произвольном месте. Выручить нас может невидимый, виртуальный объект – лунная орбита, которая является смежной и для Земли, и для Луны.

Таким образом, изображение это функция преобразования объекта в другой объект (носитель изображения) с сохранением пространственной структуры исходного объекта (оригинала).

1.2. Цветовые модели

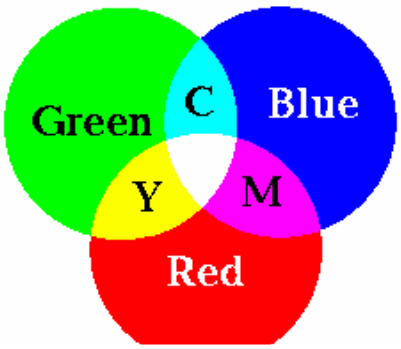


Рис.1.1. Сложение цветов

Из курса физики мы знаем, что видимый свет – это электромагнитное излучение в диапазоне длин волн от 650нм (красный свет) до 380нм (фиолетовый). Мы также знаем, что кванты света возникают при переходах электронов с высокой орбиты на более низкую. Если на множестве атомов одного и того же вещества происходят переходы на одни и те же орбиты, множество квантов имеют одинаковую энергию, и, следовательно, одну

и ту же частоту, вещество испускает монохроматический свет. Разные атомы и разные орбиты дают при излучении разные цвета. Исходя из квантовой теории света, можно сделать вывод о том, что цвет является дискретной величиной, иными словами количество различных цветов ограничено. Откуда тогда берется все многообразие цветов? Опыт подсказывает нам, что сложение цветов в различных пропорциях позволяет создавать новые цвета. Рис.1.1 показывает, что сложение красного и зеленого дает желтый цвет, красного и синего – пурпурный. Сложение красного, синего и зеленого даст белый цвет. Что же происходит при смешении разных света разных цветов? Приходится слышать, что в соответствии с волновой теорией света, при смешении электромагнитных волн разных частот происходит их наложение и интерференция. Однако, частота желтого цвета лежит между частотами красного и зеленого, а огибающая интерференционных волн намного меньше каждой из волн, участвующих в интерференции. На рис. 1.2 показано сложение двух синусоид, приблизительно пропорциональных красному и зеленому цветам.

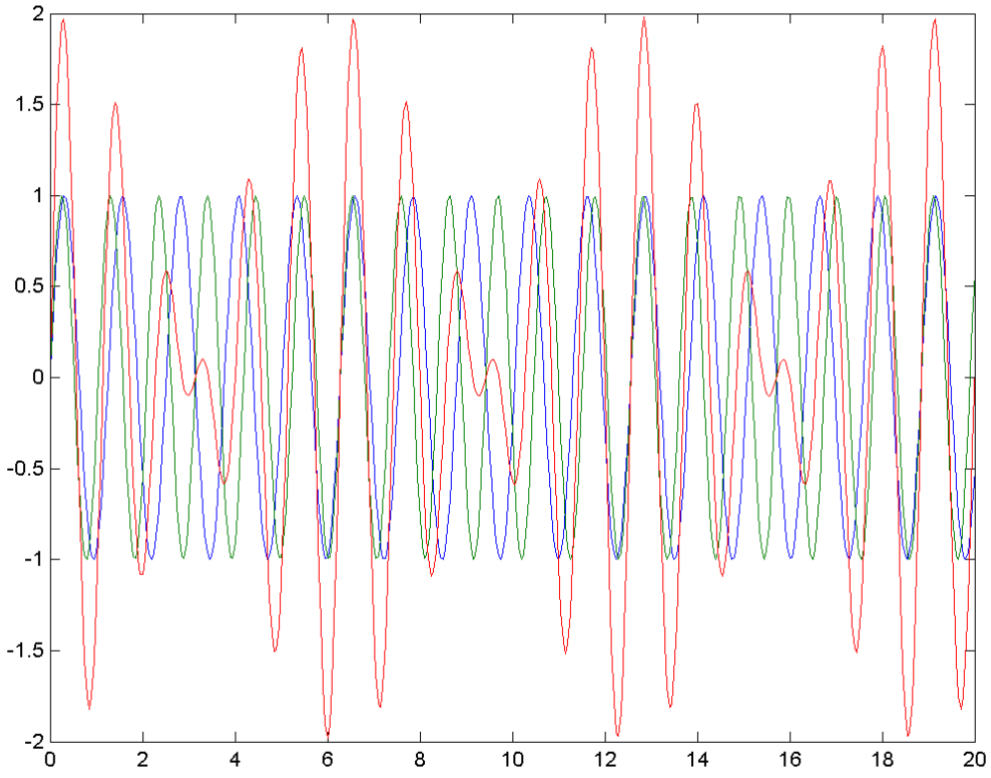


Рис.1.2. Интерференция волн красного и зеленого цветов

Как видим, ничего похожего на синусоиду с периодом, лежащим между двух исходных кривых, нет, а огибающая биений лежит далеко в области инфракрасного цвета. Другой аргумент, опровергающий интерференционную природу сложения цветов, состоит в том, что никакими комбинациями красного и синего мы не получим зеленый цвет, хотя частота зеленого цвета лежит между красным и синим так же, как желтый лежит между красным и зеленым.

Объяснение природы сложения цветов следует искать в особенностях восприятия цветов человеком. Согласно трихроматической гипотезе Юнга-Гельмгольца, сетчатка глаза человека содержит рецепторы трех цветов, красного, зеленого и синего (колбочки). Свет определенной длины волны, попадающий в глаз, вызывает возбуждение (цветовой стимул) разной интенсивности в каждом из рецепторов. Сочетание трех цветов с разными интенсивностями и дает все многообразие воспринимаемых человеком цветов, в т.ч. не существующих в природе. Например, если разложить луч белого света с помощью призмы, в получившемся спектре мы не увидим коричневого цвета. Будучи трихроматом, человек не в состоянии различить чистый цвет, например, желтый, и смесь красного и зеленого, если они дают один и тот же самый цветовой стимул.

Таким образом, для синтеза любого цвета с помощью источников света достаточно располагать источниками трех цветов, красного, зеленого и синего. Данная модель называется **аддитивной моделью синтеза цвета** или **RGB-моделью** (Red-Green-Blue).

Дневной свет, излучаемый солнцем, и который мы называем белым, состоит из множества волн различной длины, и это обстоятельство делает возможным для зрения человека различать цвета предметов, на которые это свет падает. До этого момента мы обсуждали свет, излучаемый источниками. Однако основу нашего восприятия мира с помощью зрения дает не излучаемый, а отражаемый свет, делающий видимыми все объекты материального мира, причем существенным здесь является тот факт, что не все составляющие спектра падающего на объект света отражаются. Часть спектра падающего света объект поглощает, что и обуславливает цвет, который мы видим. Если при наложении света от нескольких источников происходит сложение цветов, то при наложении красок на

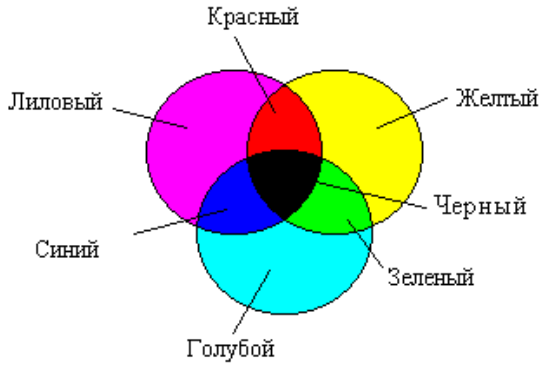


Рис.1.3. Вычитание цветов

белый фон цвета вычитаются (рис.3). Данная схема носит название **субтрактивная модель синтеза цвета**.

Синтез изображений методом вычитания существовал задолго до появления вычислительной техники в полиграфии, и здесь накоплен большой опыт в обеспечении цветопередачи. Чаще всего в качестве основных цветов используется набор **CMYK** (Cyan-Magenta-Yellow-blacK). Теоретически четвертый компонент – черный цвет – является избыточным, но используется для усиления плотности отпечатка в глубоких тенях и сокращения общего объема наносимой краски. Дело здесь не только в экономии краски, что актуально для обычных принтеров, но и в ограничении объема краски, который в состоянии принять бумага. Этот параметр бумаги называется TIL (Total Ink Limit) и должен неукоснительно соблюдаться. По этой же причине невозможно получить изображения фотографического качества на обычной бумаге для принтера.

Подготовка исходного изображения для печати называется **цветоделением** и, требует глубокого понимания полиграфических процессов. В частности, изображение, с которым работает редактор, обычно синтезируется на экране с помощью модели RGB, а конечная печать осуществляется методом CMYK. Для максимального приближения картинки на мониторе к реальной выполняется его калибровка, которая, тем не менее, не может обеспечить полное совпадение цветов в силу разных физических носителей. Существуют цвета, которые может воспроизвести монитор, но не могут быть напечатаны, и наоборот, есть цвета, которые могут быть напечатаны, но не могут быть выведены на экран [(О'Квин, 2003)].

1.3. Методы формирования изображений

Формирование изображения.

Рис.1.4. Векторный и растровый методы формирования изображений

Изображение на экране электронно-лучевой трубки (ЭЛТ или CRT – Cathode Ray Tube) формируется электронным лучом, отклоняемым с помощью магнитных полей, формируемых катушками отклоняющей системы. Возможны два способа синтеза изображений на экране (рис. 1.4). В первом случае на отклоняющую систему подаются токи, заставляющие электронный луч совершать перемещения, рисующие изображение (вектор-

ный способ). Во втором случае электронный луч совершает зигзагообразные движения по всей поверхности экрана (рисует растр) при яркости луча «чернее черного». Изображение формируется только путем управления яркостью электронного луча.

Дисплеи на основе жидкокристаллических (ЖК) матриц эмулируют развертку ЭЛТ и принципиально не могут использовать векторный способ синтеза изображений. Принтеры, использующие векторный метод, существуют и называются **графопостроители**. В графопостроителях используются пишущие элементы с краской или даже стержни от шариковых ручек, перемещаемые с помощью координатного механизма по листу бумаги.

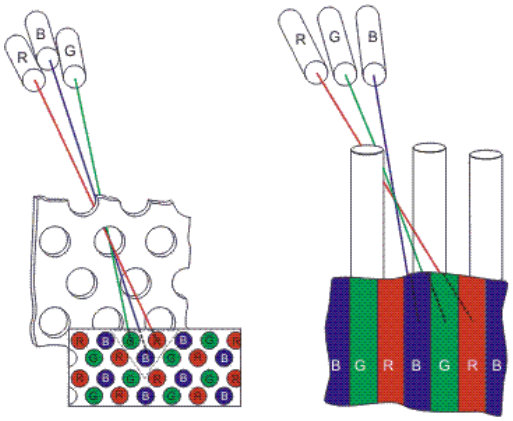


Рис.1.5. Дельта- и компланарный кинескопы

Воспроизведение цвета в дисплеях достигается смешением трех основных цветов, красного, зеленого и синего. В ЭЛТ используются три электронные пушки, расположенные треугольником (дельта-кинескоп) или в ряд (компланарный кинескоп). Несмотря на то, что электронные лучи от этих пушек называют красным, зеленым и синим, электроны, естественно, цветов не имеют. На пути лучей перед самым экраном устанавливается теневая маска с круглыми отверстиями для дельта-кинескопа или щелевыми для компланарного. С внутренней стороны экрана нанесены точки или полоски люминофора, вызывающего свечение под действием пучка электронов, причем красный луч попадает только на красный люминофор, зеленый – на зеленый и синий – на синий. Рис.1.5 (©2005-2010 Учебно-практический центр "Эксперт", www.xprt.ru) иллюстрирует принцип синтеза цвета на обоих типах кинескопов. Таким образом, каждый элемент цветного изображения состоит из **триады** – трех точек или отрезков основных цветов. Такой способ формирования объективно снижает четкость изображения и разрешающую способность цветных мониторов по сравнению с черно-белыми. При этом маска, особенно с круглыми отверстиями, существенно снижает яркость экрана. Элемент изображения на экране называется

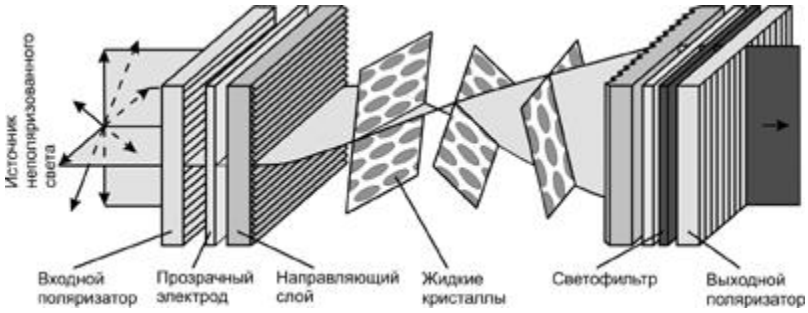


Рис.1.6. Ячейка TN-матрицы

пиксель (Pixel, Picture Element или Picture Cell), а отдельная цветная точка – **субпиксель**.

Принцип формирования изображения на ЖК мониторах схож с ЭЛТ. Каждый субпиксель состоит из светофильтра одного из цветов и жидкокристаллического затвора, пропускающего или блокирующего свет от лампы подсветки, смонтированной за матрицей. На рис. 1.6 показана схема TN (Twisted Nematic) матрицы, в которой используются закрученные нематические жидкие кристаллы (существуют нематические, смектические и холестерические жидкие кристаллы).

Рис.1.7. Ячейка TFT-матрицы

В последнее время большинство ЖК мониторов имеют активные матрицы, изготавливаемые по технологии TFT (Thin Film Transistor), отличающиеся от пассивных матриц тем, что в каждом субпикселе размещены конденсатор и тонкопленочный транзистор (1). Конденсатор хранит заряд в промежутке между обновлениями экрана, а транзистор поддерживает напряжение, приложенное к слою жидких кристаллов, снижая его мерцание (рис.1.7). Подсветка матрицы осуществляется либо люминесцентной лампой, либо светодиодами. Светодиодная подсветка позволяет существенно уменьшить толщину монитора, а также повысить контрастность. Для этого подсветка (обычно один светодиод подсвечивает несколько смежных пикселей) выключается в местах экрана, где должен быть черный цвет.

Таким образом, изображение на экране дисплея формируется из пикселей, каждый из которых состоит из триады – субпикселей трех основных цветов. При этом пикселем иногда называют минимальный элемент изображения, который может быть выведен на экран. Эти два понятия совпадают только для ЖК монитора и только в режиме максимального разрешения. В режимах, отличных от максимального разрешения, точка, вводимая на экран, занимает несколько пикселей, а в ЭЛТ дисплеях размер выводимой точки ограничивается еще и фокусировкой луча.

ГЛАВА 2. РАБОТА ГРАФИЧЕСКОГО АДАПТЕРА

2.1. Принципы телевизионной развертки

Вывод изображения на экран компьютерного монитора построен на тех же принципах, на которых работает телевидение. На строчные и кадровые катушки и отклоняющей системы подаются пилообразные напряжения таким образом, что электронный луч пробегает по горизонтали столько раз, сколько строк в растре, постоянно смещаясь по вертикали вниз. После завершения отображения кадра луч резко возвращается вверх, и начинает отображаться следующий кадр. Изображение на экране обновляется 25 раз в секунду, что обусловлено свойствами человеческого зрения: смена кадров при такой скорости происходит незаметно для глаза, и человек видит вместо последовательности кадров непрерывное движение. На скорости 24 кадра в секунду сняты все фильмы начиная с 20-х годов прошлого века. Кстати, предыдущий стандарт в кино был 16 кадров в секунду, чем и объясняется тот факт, что кинохроники начала 20-го века показывают все происходящее в ускоренном темпе. Тем не менее, мерцание экрана при скорости 25 раз в секунду на экране ЭЛТ все же заметно. Причина в том, что на киноэкране большую часть времени отображается весь кадр, а на телевизионном в каждый момент светится только одна точка, в которой находится электронный луч. Весь кадр виден нам только за счет небольшого послесвечения люминофора и инерции зрения.

Рис.2.1. Прогрессивная развертка

Решение, позволяющее устранить мерцание при сохранении частоты кадров 25 в секунду, было изобретено одновременно с первым телевизором. Это **чересстрочная развертка**, заключающаяся в том, что электронный луч пробегает весь экран 50 раз в секунду, но первый кадр содержит только нечетные строки, а второй – четные. Может возникнуть вопрос, почему в телевидении 25 кадров в секунду, если в кино 24? Раньше эффект от этой разницы можно наблюдать в старых фильмах на киноэкране: если в кадр попадал работающий телевизор, то на его экране каждую секунду пробегала черная полоса, а изображение постоянно смещалось вниз. Здесь объяснение простое: в нашей стране

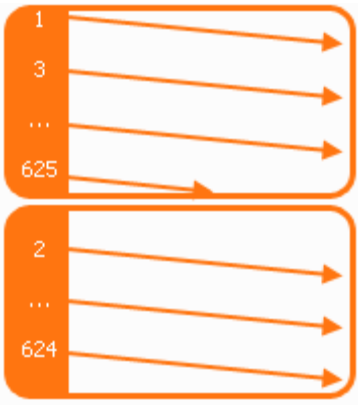


Рис.2.2. Чересстрочная развертка

частота переменного тока 50гц, и если сделать частоту строк 48 полукадров, то помехи от электросети неизбежно будут попадать на экран. Теоретически сеть переменного тока можно использовать и для синхронизации телевизионной развертки, однако, это возможно только в пределах единой энергосистемы и только в прямом эфире, когда луч в передающей ЭЛТ (иконоскопе) на телецентре и лучи в кинескопах всех телевизоров движутся синхронно с генераторами всех электростанций. В этой связи в составе телевизионного сигнала есть срочные и кадровые синхроимпульсы, которые позволяют синхронизировать развертку. Кстати, в современных мониторах имеется режим развертки 60 кадров в секунду, имеющий происхождение из стран, в которых частота переменного тока составляет 60гц. Мониторы на ЭЛТ последних моделей имеют обычную, а не чересстрочную развертку, которая называется **прогрессивной разверткой.**

2.2. Зачем нужен видеоадаптер

Для вывода на экран ЭЛТ изображение должно быть представлено в виде трех последовательностей аналоговых сигналов RGB, управляющих соответствующими электронными пушками. ЖК мониторы могут иметь цифровой вход, через который подаются те же сигналы RGB, правда, уже в цифровой форме. И в том, и в другом случае существует задача регенерации изображения. Даже если картинка на экране статичная, монитор должен непрерывно получать данные, объем которых достаточно большой. Если возложить задачу регенерации изображения на центральный процессор, то на другие задачи у него просто не останется времени, тем более, что эта задача должна иметь наивысший приоритет, иначе картинка на экране будет все время сбиваться.

Следовательно, необходим **видеоконтроллер** или **видеоадаптер** или **видеокарта** – устройство, которое возьмет на себя функцию преобразования изображения, получаемого от программы, в последовательность видеосигналов. Программа, формирующая изображение, должна передавать информацию однократно, после чего видеоадаптер будет регенерировать изображение с частотой развертки. Следовательно, образ экрана должен храниться в видеоадаптере, для чего последний должен располагать **видеопамятью.**

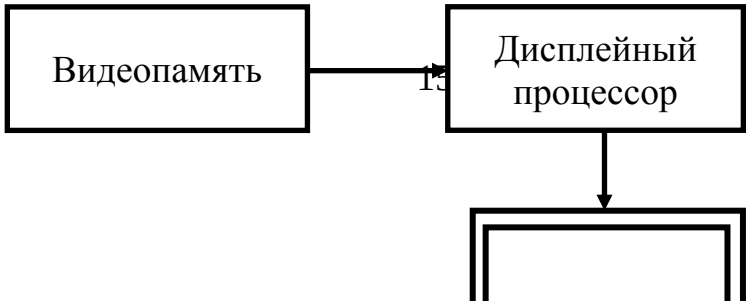


Рис.2.3. Структура видеоадаптера

Другая проблема – скорость обмена данными между программой и видеопамятью. Видеоадаптер подключается к системной шине компьютера как периферийное устройство, и обмен данными с ним осуществляется с помощью команд ввода-вывода, причем, сначала надо указать видеоадаптеру адрес видеопамяти, затем передать байт данных. Таким образом, отображение даже одной точки на экране потребует около десятка команд процессора и не менее двух операций ввода-вывода, а длительность заполнения всего экрана будет такой, что и речи не может быть об отображении подвижных объектов.

Решение проблемы в архитектуре IBM PC заключается в том, что видеопамять, хотя и размещается физически на плате видеоадаптера, но находится в общем адресном пространстве с основной памятью. Таким образом, программа осуществляет обмен данными с видеопамятью обычными командами процессора, а протокол обмена данными с видеопамятью не отличается от обмена с оперативной памятью, которая также подключена к системной шине.

2.3. Организация видеопамяти и режимы видеоадаптера

Организацию видеопамяти рассмотрим на примере семейства видеоадаптеров CGA, EGA и VGA, использовавшихся в ранних моделях IBM PC XT/AT. Следует заметить, что каждый из видеоадаптеров может работать в нескольких режимах, причем более поздние модели поддерживают все режимы ранее разработанных видеоадаптеров, что позволяет обеспечить совместимость программного обеспечения сверху вниз: старые программы могут запускаться на новых компьютерах.

Технические решения, использованные в разных моделях адаптеров, могут показаться нелогичными, но следует иметь в виду, что они разрабатывались для первых **персональных** компьютеров, т.е. в условиях жестких ограничений на стоимость и габариты. Кроме того, разработчики архитектуры видеоподсистем, как, впрочем, и проекта IBM PC в целом,

явно не верили в Закон Мура (2), согласно которому число транзисторов на кристалле удваивается каждые полтора-два года.

Черно-белый режим адаптера CGA (Color Graphic Adapter), устанавливавшегося на ПК IBM PC XT, имеет разрешение 640x200 битовых точек. Организация памяти в режиме CGA максимально приспособлена для чересстрочной развертки. С сегментного адреса B800h начинается банк четных строк, а с адреса BA00h – банк нечетных строк. Нумерация строк начинается с нуля, поэтому такое расположение банков памяти является логичным. Каждой точке на экране соответствует один бит. Таким образом, каждая строка занимает 80 байт, а четный полукадр – 8000 байт. Нечетный полукадр находится со смещением 2000h = 8192. Лишние 192 байта не используются. Такое решение продиктовано соображением простоты адресации.

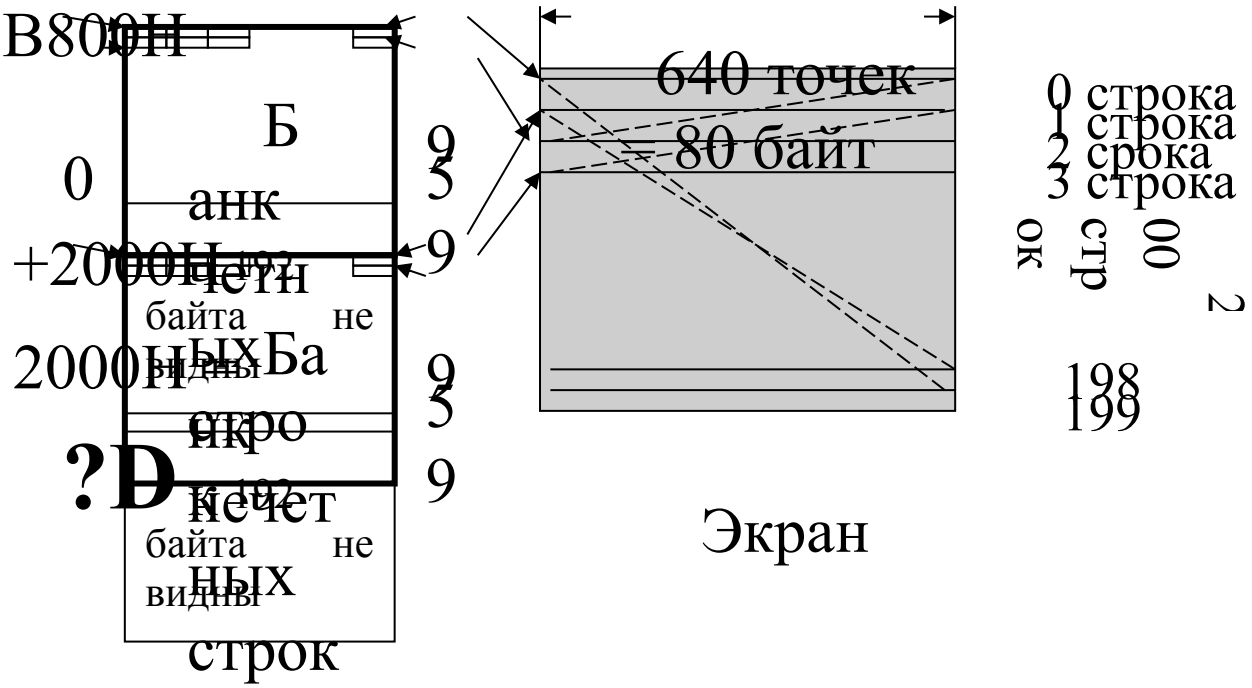


Рис. 2.4. Видеопамять CGA в черно-белом режиме

Цветной режим адаптера CGA отличается от монохромного тем, что на каждую точку отводится два бита, а разрешение экрана составляет 320x200 точек. Два бита на точку дают 4 цвета, а какие должны быть цвета, определяется **палитрой**. Палитра – это набор регистров, содержащих коды цветов, а код точки в видеопамяти – номер регистра палитры. Таким образом, количество различных цветов, отображаемых на экране, может быть очень большим, но одновременно на экране не может быть больше числа комбинаций кода точки, в данном случае не больше четырех.

Адаптер CGA был первым графическим видеоадаптером в семействе

IBM PC, и основным назначением графического режима было отображение графиков в программах научных расчетов. Для этих целей монохромный режим или четыре цвета были вполне достаточными, а вычислительная мощность первых ПК не позволяла создавать программы с более продвинутой графикой.

В более мощной модели IBM PC AT устанавливался видеоадаптер EGA (Enhanced Graphic Adapter), имеющий такое же разрешение, как в CGA, но 16 цветов. Для отображения 16 цветов каждая точка должна быть представлена четырьмя битами. Видимо, с целью экономии адресного пространства, а архитектура IBM PC не позволяла адресовать более 1Мб оперативной памяти, разработчики адаптера EGA разбили видеопамять на четыре плоскости, каждая из которых начинается с одного и того же адреса (рис. 2.5). Каждый бит адресном пространстве видеопамяти соответствует точке на экране. Три плоскости соответствуют основным цветам RGB, а четвертая плоскость хранит атрибут яркости.

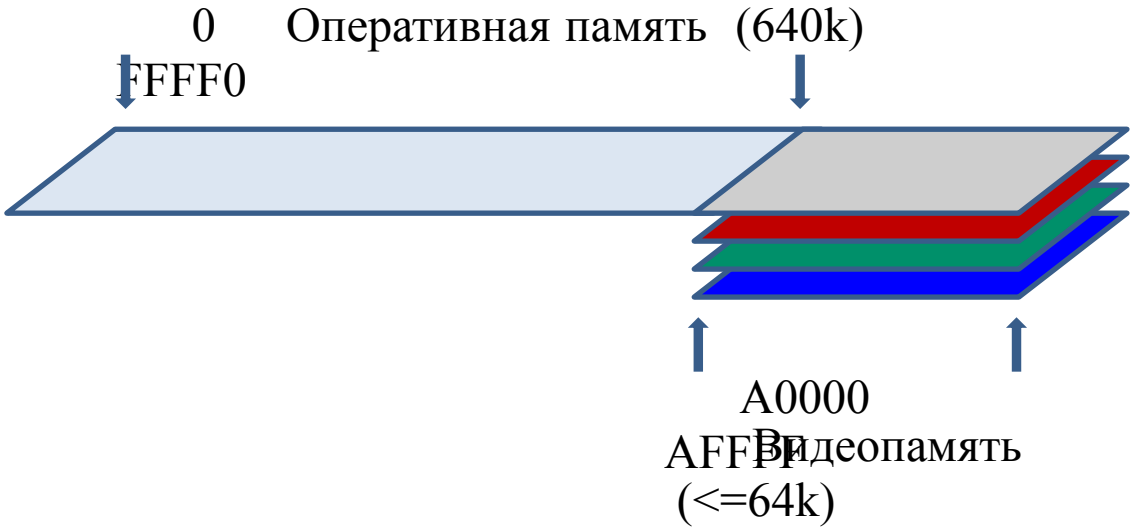


Рис. 2.5. Видеопамять EGA

При записи байта в видеопамять происходит запись сразу во все плоскости, при чтении – читается байт из нулевой плоскости, отвечающий за синий цвет (установлено опытным путем, в описаниях адаптера этой информации нет). Параллельная запись во все четыре слоя означает, что все точки отображаются в белом цвете. Если требуется вывод точек другого цвета, то должен быть установлен регистр маски цвета. Другая проблема – каждой точке на экране соответствует один бит в каждой видеоплоскости, а записать в память можно не меньше байта. Для маскировки битов, которые не должны меняться, также используется маска (рис.2.6). Более подробно это обсуждается в главе 3.

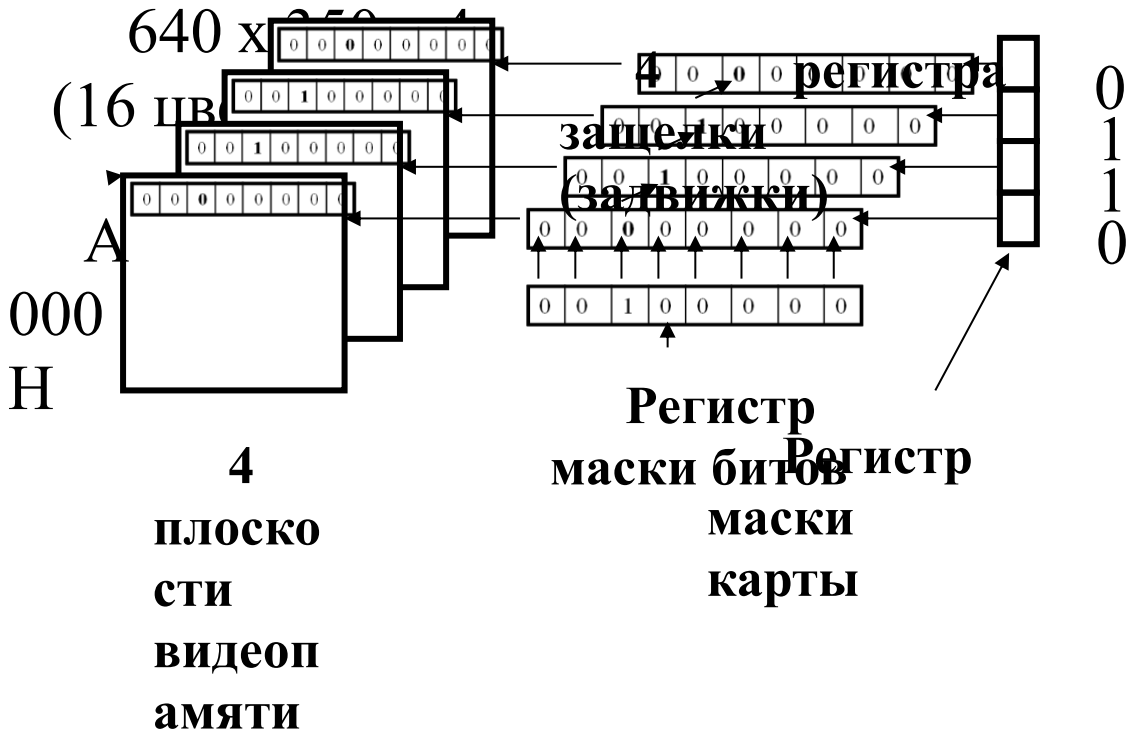


Рис.2.6. Организация доступа к видеопамяти видеоадаптера EGA.

Следующим поколением видеоадаптеров стал адаптер VGA, который в самом простом режиме обеспечивает разрешение 320x200 пикселей и 256 цветов, а в самом продвинутом – 1024x768. Разные модели видеокарт имели разный объем видеопамяти, от 64кб до 1Мб. При увеличении объема видеопамяти возникает проблема адресного пространства. В архитектуре IBM PC для видеопамяти отведено только 64кб адресов. При разрешении 640x480 и ниже на каждый цвет можно было отвести отдельную плоскость в видеопамяти, увеличивая число плоскостей по мере возрастания числа цветов. Но при дальнейшем увеличении разрешающей способности экрана адресного пространства уже не хватит даже по биту на пиксель. В адаптере VGA каждый пиксель представлен целым байтом в видеопамяти, а проблема адресации решается тем, что адресное пространство видеопамяти указывает на окно в видеопамяти размером в 64кб, а специальный регистр в видеоконтроллере содержит адрес окна. Таким образом, чтобы заполнить весь экран, необходимо несколько раз сдвигать окно.

Адаптер SVGA (SuperVGA) стал эволюционным продолжением архитектуры VGA и обеспечивает разрешение 800x600, 1024x768, 1280x1024. Цвет может кодироваться числом бит 8, 16 или 24, что соответствует 256, 64к или 16,7 млн. цветов.

2.4. Кодирование цветов

Как было рассмотрено раньше, любой цвет может быть воспроизведен с помощью смешивания трех цветов, например, RGB или CMY. Число три плохо накладывается на байтовую организацию памяти ЭВМ, и в ходе эволюции видеорежимов число бит на пиксель увеличивалось удвоением: 1, 2, 4, 8, 16, 32. Дальнейшее увеличение глубины цвета не имеет смысла, поскольку нет технических средств, позволяющих отобразить столько цветов, и человек не в состоянии их различить.

В режимах видеоадаптеров с четырьмя битами на цвет лишний бит использован для задания двух уровней яркости всех компонентов пикселя. Таким образом, восемь комбинаций цветов RGB в сочетании с атрибутом яркости дают 16 разных цветов:

Таблица 2.1. Коды цветов EGA

Яркость	Красный	Зеленый	Синий	Цвет
0	0	0	0	Черный
0	0	0	1	Синий
0	0	1	0	Зеленый
0	0	1	1	Циан
0	1	0	0	Красный
0	1	0	1	Фиолетовый
0	1	1	0	Коричневый
0	1	1	1	Серый
1	0	0	0	Темно-серый
1	0	0	1	Ярко-синий
1	0	1	0	Ярко-зеленый
1	0	1	1	Яркий циан
1	1	0	0	Ярко-красный
1	1	0	1	Пурпурный
1	1	1	0	Желтый
1	1	1	1	Белый

Таким образом, каждый цвет кодируется двумя битами, один из которых меняется независимо, а второй устанавливается для всех цветов одинаково.

В 256-цветном режиме VGA нет битов, отвечающих за красный, зеленый или синий цвета. В составе видеоконтроллера имеются 256 регистров ЦАП (цифро-аналоговых преобразователей), которые преобразуют 8-битные данные в три 6-битных сигнала для ЦАП (рис.2.6).

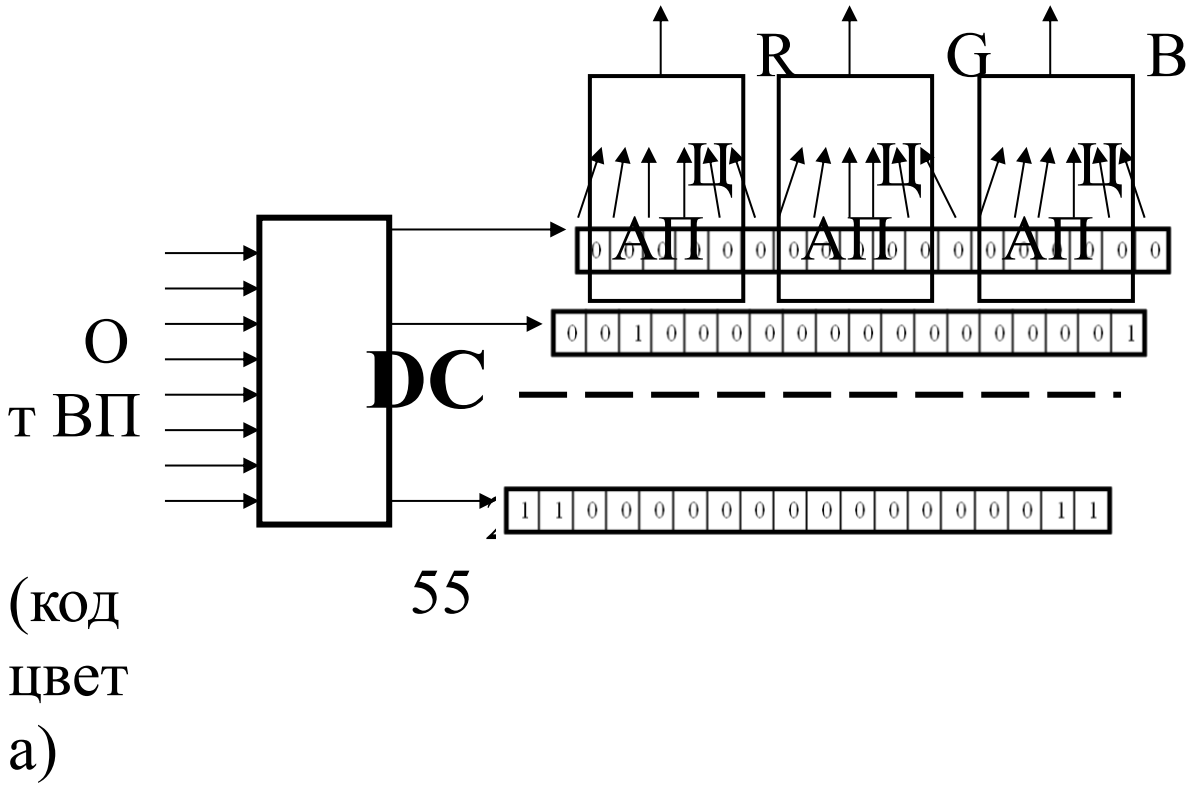


Рис.2.6. Декодирование цвета в адаптере VGA

Для режимов с бóльшей глубиной цвета такой подход не может использоваться, поскольку потребуется 2^{16} = 65536 регистров и более. В таких режимах используется прямое кодирование цвета (Direct Color Mode). Два варианта кодирования цвета в адаптере Diamond Stealth 64 представлены на рис. 2.7.

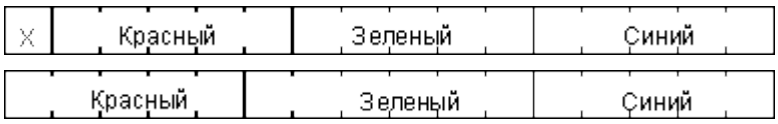


Рис.2.7. Кодирование цветов с глубиной 15 и 16 бит.

В первом случае старший бит не используется, а во втором – увеличена глубина зеленого, поскольку чувствительность человеческого глаза к зеленому цвету выше, чем к красному или синему.

При глубине цвета 24 бита для каждого пикселя в видеопамяти могут отводиться три либо четыре бита; во втором случае четвертый байт является резервным.

2.5. Режимы работы видеоадаптеров

Как было сказано выше, графический режим видеоадаптера и модель видеоадаптера не одно и то же. Каждый адаптер имеет свои собственные видеорежимы и при этом вынужден поддерживать режимы всех предыдущих моделей для обеспечения программной совместимости компьютеров. В табл. 2.2 приведены некоторые режимы работы видеоадаптеров.

Таблица 2.2. Режимы работы видеоадаптеров

Режим	Поддерживающие адаптеры	Разрешение	Количество
4, 5	CGA, EGA, VGA, SVGA	320x200	4
6	CGA, EGA, VGA, SVGA	640x200	2
0Dh	EGA, VGA, SVGA	320x200	16
0Eh	EGA, VGA, SVGA	640x200	16
10h	EGA, VGA, SVGA	640x350	16
12h	EGA, VGA, SVGA	640x480	16
13h	VGA, SVGA	320x200	256
101h	VGA, SVGA	640x480	256
103h	VGA, SVGA	800x600	256
107h	VGA, SVGA	1280x1024	256
117h	SVGA	1024x768	65535
118h	VGA, SVGA	1024x768	16777216

Поскольку конкретный видеоадаптер может поддерживать несколько видеорежимов, программа должна ему сообщить, какой видеорежим требуется, т.е. загрузить код режима в соответствующий регистр видеоконтроллера. В операционной системе MS DOS для этого используется прерывание 10h BIOS:

```
MOV     AH,0    ; функция BIOS: Установить видеорежим
MOV     AL,0Dh  ; Код режима 0Dh – 320x200x16
INT     10H     ; Прерывание BIOS
```

Необходимо заметить, что видеоадаптер не имеет обратной связи, и прерывание 10h не возвращает результата. Это означает, что ошибка установки видеорежима может привести к тому, что отображение информации на экране будет неправильным, и, чтобы вернуться в исходное состояние, придется перезагрузить компьютер. Именно поэтому в ОС Windows при изменении установок экрана требуется подтвердить, что этот режим был правильно установлен. Если пользователь не дает подтверждения, программа возвращает режим видеоадаптера в исходное

состояние.

ГЛАВА 3. ОСНОВЫ ПРОГРАММИРОВАНИЯ ВИДЕОАДАПТЕРОВ

3.1. Особенности адресации видеопамяти

В подразд. 2.3 говорилось о том, что для ускорения доступа видеопамять размещается в адресном пространстве оперативной памяти, хотя физически находится на плате видеокарты. Единственным ограничением является то, что для адресации с помощью пары регистров может использоваться только регистр BX. Для адресации начала видеобуфера может использоваться следующая последовательность команд:

```
MOV     AX, A000h  ; адрес видеопамяти
MOV     ES, AX     ; устанавливаем сегментный регистр
MOV     BX, 0      ; смещение (начало видеопамяти)
```

Рассмотрим приемы обращения к видеопамяти с битовой организацией (один бит видеопамяти соответствует точке на экране). Если после этого выполнить команду

```
MOV     ES, [BX], 10000000b
```

то в верхнем левом углу экрана отобразится белая точка, а если быть точнее, то одна белая и семь черных точек. Допускается непосредственное указание смещения и использование индексных регистров. Как и при обычном обращении к памяти содержимое двух байтов 16-разрядного регистра в памяти поменяется местами:

```
MOV     ES:[BX], 00FFh
```

При выполнении этой команды на экране будет отображено сначала 8 белых, затем 8 черных точек. Работают и цепочечные команды. Так, залить экран белым цветом можно такой последовательностью команд:

```
XOR  DI, DI          ; обнуляем индексный регистр
MOV CX, 0FFFFh       ; счетчик цепочки команд
MOV AL, 0Fh          ; заносим код белого цвета
REP   STOSB          ; выполнить цепочку записи байта
```

Таким путем можно вывести на экран изображение целиком, причем только белым цветом, т.е. записью во все плоскости сразу. Чтобы обращаться к плоскостям видеопамяти по отдельности, нужна еще одна координата – номер плоскости. Однако, удобнее вместо раздельной записи в разные плоскости иметь возможность записывать данные сразу в несколько, чтобы получить нужный цвет. Такую возможность дает маска

цвета, которая размещается в видеоконтроллере. Для обращения к маске используется пара адресов основной шины: 3C4h и 3C5h. Поскольку по данным адресам находится несколько регистров, первый из этих адресов используется для указания индекса регистра (маска цвета имеет индекс 2), второй – для передачи данных (самой маски). Ниже приведена последовательность команд, с помощью которой устанавливается код цвета 07h (серый цвет).

```
MOV  DX, 3C4h      ; указатель на адресный регистр
MOV  AL, 2         ; индекс регистра маски цвета
OUT  DX, AL        ; устанавливаем адрес регистра маски
INC  DX            ; адрес регистра данных
MOV  AL, 07h       ; код цвета (серый)
OUT  DX, AL        ; посылаем код цвета в контроллер
```

Таким образом, если мы хотим отобразить точку определенного цвета, то мы должны предварительно установить маску, которая разрешит запись только в плоскости, для которой бит маски установлен в единицу. Назначение битов маски цвета приведено в табл.3.1.

Таблица 3.1. Маска цвета

Бит маски	Назначение
7	Не используется
6	
5	
4	
3	Яркость
2	Красный
1	Зеленый
0	Синий

Следует заметить, что при записи в видеопамять соответствующие биты в замаскированных плоскостях не заполняются нулями, а сохраняют прежние значения, в результате чего происходит смешение цветом. Предположим, мы хотим записать точку ярко-красного цвета, код которой в соответствии с табл. 2.1 равен 1100b, а текущий цвет данной точки – серый (код 0111b). Маска цвета разрешит запись в две плоскости (яркость и красный цвет), но при этом замаскированные плоскости зеленого и синего цветов уже имеют единицы. В результате вместо ярко-красного мы получим белый цвет, как показано в табл. 3.2.

Таблица 3.2.

Код цвета	Маска цвета	Цвет точки в памяти	Результат
Яркость	1	0	1
Красный	1	1	1

Зеленый	0	1	1
Синий	0	1	1

Обозначим булевыми переменными b – бит, записываемый в память; o – бит, хранящийся в памяти; m – значение бита маски для некоторй плоскости. Тогда значение n бита в данной плоскости видеопамяти после записи будет определяться следующим образом:

$$n = b \wedge m \wedge o \vee b \wedge \bar{m} \wedge o \vee b \wedge m \wedge \bar{o} \vee \bar{b} \wedge \bar{m} \wedge o =$$
$$= b \wedge (o \vee m) \vee \bar{b} \wedge \bar{m} \wedge o$$

Таким образом, чтобы гарантированно отобразить точку нужного цвета нужно сначала установить маску цвета 1111b (две команды OUT) и записать в нужное место экрана нуль, затем уже установить нужный цвет (еще две команды OUT) и записать его в видеопамять. Очевидно, что такой путь сводит на нет ускорение доступа к видеопамяти с помощью прямой адресации, и программирование графики в режимах EGA является в определенной степени искусством. Например, если известно, что фон рисунка уже залит серым цветом (код 0111b), то рисовать на этом фоне синие линии можно путем установки маску 1110b и записи нулей в соответствующие места на экране.

3.2. Маскирование битов и вычисление адреса точки

Ранее мы говорили о том, что система процессор – память не может обмениваться отдельными битами. Минимальной единицей данных здесь является байт, и в случае битовой организации видеопамяти, записывая байт с одной единицей и остальными нулями, мы обнуляем соседние точки. Избежать этого позволяет такой же механизм, как и при выборе плоскостей видеопамяти – маскирование. Битовая маска располагается в восьмом регистре видеоконтроллера, доступ к которым осуществляется аналогично маске цвета парой портов с адресами 3CEh и 3CFh.

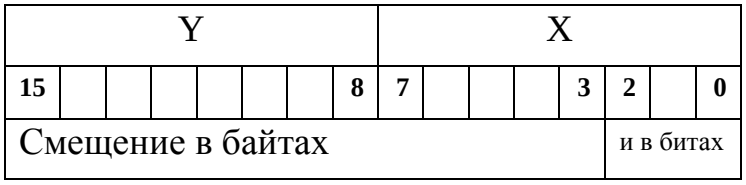
Предположим, мы хотим отобразить на экране одну точку со смещением 2 от начала байта, оставив соседние точки без изменений. Тогда мы должны установить значение битовой маски 00100000b с помощью следующего фрагмента кода:

```
MOV  DX, 3CEh        ; указатель на адресный регистр
MOV  AL, 8           ; индекс регистра битовой маски
OUT  DX, AL          ; устанавливаем адрес регистра маски
INC  DX              ; адрес регистра данных
MOV  AL, 20h         ; битовая маска 00100000
OUT  DX, AL          ; посылаем маску в контроллер
```

Таким образом, битовая маска и маска цвета совместно определяют, в какие разряды и в какие плоскости должны записываться данные.

Кроме проблем записи битовая организация видеопамяти влечет за собой также нетривиальную задачу вычисления адреса видеопамяти для каждой точки. При создании изображения обычно оперируют координатами точек по горизонтали и вертикали (*X* и *Y*). Поскольку развертка ЭЛТ монитора делается сверху вниз, видеопамять также начинается с верхнего левого угла экрана, т.е. в привычном понимании горизонтальная ось *X* идет слева направо, а вертикальная ось *Y* – сверху вниз. Чтобы вывести точку на экран, надо преобразовать координаты *X* и *Y* в смещение относительно начала видеобуфера в виде пары значений – номера байта и смещения от начала байта.

Рассмотрим для начала гипотетический режим экрана 256x256 точек, который удобен тем, что координаты *X* и *Y* занимают по целому байту. В этом случае 16-разрядное число



это номер бита от начала видеопамяти, а его старшие 12 разрядов – порядковый номер байта, а младшие 3 разряда – смещение внутри байта.

Ниже показан фрагмент кода, с помощью которого можно преобразовать координаты в соответствующие смещения. В результате выполнения данного фрагмента в регистре BX будет байтовое смещение, а в регистре DL – битовое (внутри байта).

```
X  DB    ?            ; координата X
Y  DB    ?            ; координата Y
   …
   MOV   BH, Y        ; загружаем координаты X
   MOV   BL, X        ; и Y
   MOV   DL, BL       ; перегружаем X в DL
   MOV   CL, 3        ; счетчик сдвига = 3
   SHR   BX, CL       ; деление на 8 логическим сдвигом
   AND   DL, 03h      ; находим битовое смещение
```

Следующая задача – преобразовать двоичное смещение внутри байта в битовую маску, т.е. выполнить дешифрацию. Самый очевидный способ

– табличный:

```
T   DB    128,64,32,16,8,4,2,1      ; таблица
    …
    XOR   DH, DH          ; очищаем старший байт регистра DX
    MOV   SI, DX          ; загружаем индексный регистр
    MOV   AL, T+[SI]      ; загружаем маску в AL
```

Второй способ преобразовать двоичный код в битовую маску – командой сдвига вправо, используя этот код в качестве счетчика сдвигов и исходное значение маски 10000000b.

Вычисление адреса точки в видеорежимах с другими разрешениями экрана не может быть выполнено столь же изящно, поскольку координаты не занимают целые байты. Тем не менее, число точек по горизонтали всегда кратно восьми, так что, по меньшей мере, каждая строка начинается с начала байта.

Рассмотрим, как преобразовать координаты в адрес точки при длине строки, равной 640 пикселей, что соответствует длине буфера 80 байтов на каждую строку. Число строк нас при этом не тревожит, т.к. в любом случае оно укладывается в два байта. Для нас важнее тот факт, что координата *X* занимает 10 бит.

Y																**X**															
15															**0**	**15**					**10**	**9**						**3**	**2**		**0**
Смещение по вертикали																**Не используется**						**Смещение по строке байтовое**							**… и битовое**		

Как и в предыдущем примере, три младших разряда координаты *X* соответствуют битовому смещению, а следующие семь разрядов вместе с координатой *Y* определяют байтовое смещение.

Десятичное число 80 не является степенью двойки, поэтому мы не можем просто присоединить справа к координате *Y* семь разрядов координаты *X*. Смещение в байтах *S* определяется следующей формулой:

$$S = Y * 80 + X \text{ div } 8.$$

Поскольку 80 = 64 + 16, вычислить *S* без использования операций деления и умножения можно с помощью сдвигов:

```
X  DW    ?              ; координата X
Y  DW    ?              ; координата Y
   …
   MOV   AX, Y          ; загружаем координаты Y
   MOV   BX, X          ; и X
   MOV   DX, BX         ; перегружаем X в DX
   MOV   CL, 3          ; счетчик сдвига = 3
   SHR   BX, CL         ; деление на 8 логическим сдвигом
   AND   DL, 07h        ; находим битовое смещение
   MOV   CL, 4          ; счетчик сдвига = 4
   SAL   AX, CL         ; сдвиг влево на 4 == умножение на 16
   ADD   BX, AX         ; прибавляем к смещению Y
   MOV   CL, 2          ; счетчик сдвига = 2
   SAL   AX, CL         ; сдвиг влево  2 == умножение еще на 4
   ADD   BX, AX         ; прибавляем к смещению X
```

После выполнения данного кода в регистре BX будет смещение точки от начала видеобуфера в байтах, а в DL – в битах от начала байта. Может возникнуть вопрос, почему мы избегаем операций умножения. Объяснение здесь простое: Во-первых, при формировании сложного изображения число таких операций может измеряться многими тысячами, что сделает невозможным динамическое отображение в реальном масштабе времени. Во-вторых, аппаратная реализация операций с ограниченным набором команд гораздо проще.

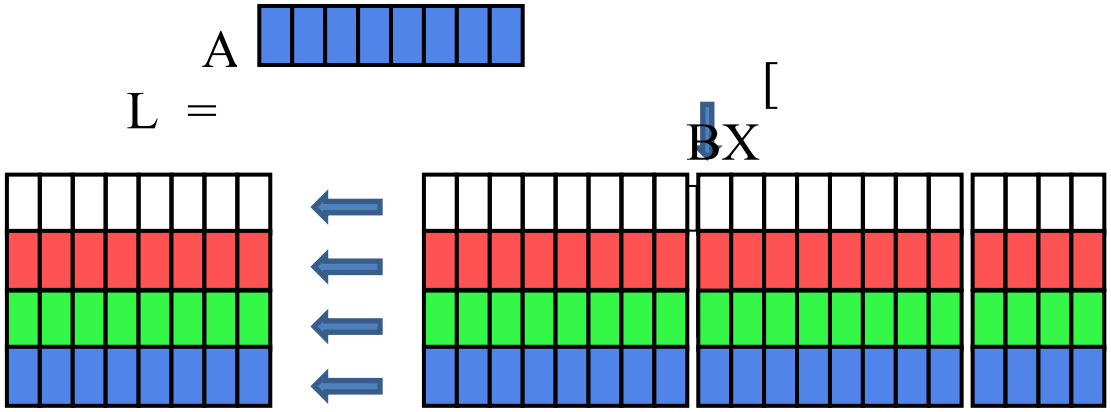
3.3. Операции с изображениями внутри видеоадаптера

Ранее мы убедились, в том, что результат записи данных в видеопамять может давать непредсказуемые результаты, если программа игнорирует текущее состояние памяти. Также мы видели, что при битовой организации памяти мы можем записывать данные в видеобуфер как одновременно во все биты и плоскости, так и избирательно, с использованием масок. Однако чтение данных из видеобуфера позволяет обращаться только к нулевой плоскости, соответствующей синему цвету. Между тем, задачи отображения часто требуют знания кодов цветов, ранее записанных в видеопамять, например, для изображения теней, отбрасываемых объектом на фон, или для замены одного цвета на другой в определенной области видеопамяти.

Адаптер EGA предоставляет возможность извлекать байты из всех плоскостей видеопамяти во внутренние регистры-защелки (*latch registers*). Содержимое этих регистров недоступно программе, но оно может участвовать в логических операциях над данными, заносимыми в видеопамять. При каждом чтении байта из видеопамяти и записи в

видеопамять данные из всех четырех плоскостей записываются в четыре восьмиразрядных регистра-защелки:

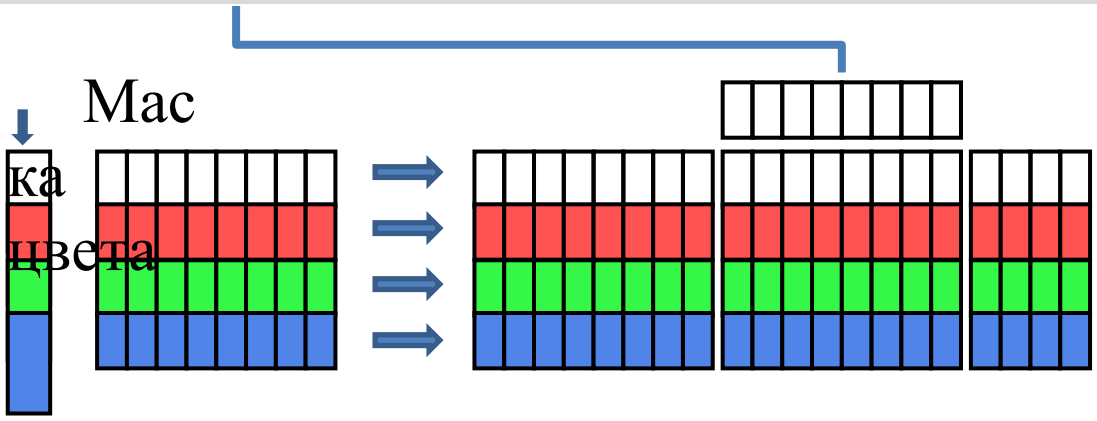
```
MOV     AL, ES:[BX]          ; загрузка синего байта
```



Заметим, что после вышеописанной операции содержимое регистра AL обычно лишено смысла, поскольку отражает только одну составляющую цвета. Цель этой операции – загрузка регистров-защелок из указанного места видеопамяти. Увидеть содержимое этих регистров программа не может, но при записи в видеопамять может записываться не байт, указываемый в команде, а результат логической операции этого байта и регистра-защелки. Допускаются следующие операции: AND (И), OR (ИЛИ), XOR (Исключающее ИЛИ).

Ниже приведен пример кода, который записывает точку в крайний левый бит байта со смещением [BX] при условии, что в регистре …установлено значение 10 (OR).

```
MOV  AL, ES:[BX]          ;  читаем байт из видеопамяти
MOV  AL, 10000000b        ;  точка в первом бите
MOV  ES:[BX], AL          ;  записываем байт в видеопамять
```



В этом примере делается запись точки желтого цвета (маска цвета = 0Eh), но в старших битах регистров-защелок был записан фиолетовый цвет (код 05h). Поскольку 0Eh ∨ 05h = 0Fh, вместо желтой точки в указанное место будет записана белая.

Пусть теперь регистр …имеет значение 01 (AND). Тот же самый пример даст в результате 0Eh ∧ 05h = 04h, т.е. будет записана красная точка. Наконец, если регистр …установлен в 11 (XOR), результат будет 0Eh XOR 05h = 0Bh (яркий циан).

Одно из основных назначений регистров-защелок – заливка двумерных фигур однородным цветом – будет рассмотрено позже. В ходе этой операции необходимо выявлять границы закрашиваемой области. Для этого используется операция сравнения …

ГЛАВА 4. ПОСТРОЕНИЕ ЛИНИЙ НА РАСТРЕ

4.1. Построение горизонтальных и вертикальных линий

Владея алгоритмами построения точки на растре, можно реализовать их в виде макросов или процедур и по точкам рисовать горизонтальные и вертикальные линии. Однако этот путь не самый эффективный, что особенно заметно на горизонтальных линиях.

Рассмотрим алгоритм рисования вертикальной линии. Для нее сначала должны быть вычислены смещения начала и конца отрезка в целых байтах S1 и S2, а также общее для всех точек битовое смещение B.

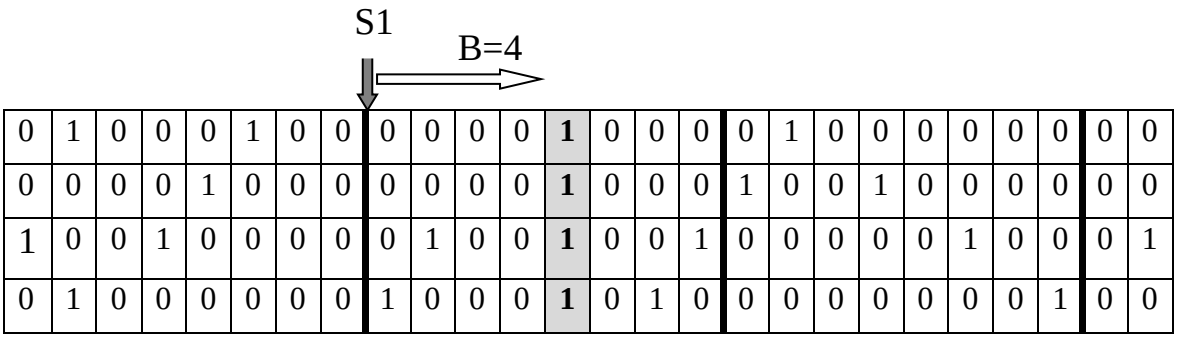


Рис.4.1. Расположение вертикальной линии на битовом растре.

Алгоритм построения вертикального отрезка прямой, заданного одной координатой *X* (смещение от левого края экрана) и двумя координатами *Y1* и *Y2* концов отрезка может быть следующим.

1. Вычислить смещения начала и конца отрезка в видеобуфере S1 и S2. Вычислить битовое смещение B.
2. Установить атрибуты точки, в т.ч. битовую маску.
3. Вывести на экран точку со смещением S1.
4. Увеличить смещение S1 на величину числа байт в одной строке.
5. Если S1 меньше или равно S2, повторить с п.3, иначе конец.

Построение горизонтального отрезка прямой при байтовой (или более) организации видеопамяти еще проще, поскольку можно использовать цепочечные команды:

```
S1      DW   ?                ; смещение первой точки
S2      DW   ?                ; смещение второй точки
        …
        MOV DI, S1            ; устанавливаем смещение на начало
        MOV CX, S2            ; находим
        SUB  CX, S1           ;          длину отрезка
        MOV AL, 00Eh          ; будем заполнять желтым цветом
        REP  STOSB
```

Несколько сложнее рисовать горизонтальную линию на битовом растре. На рис. 4.2 приведены три варианта расположения точек.

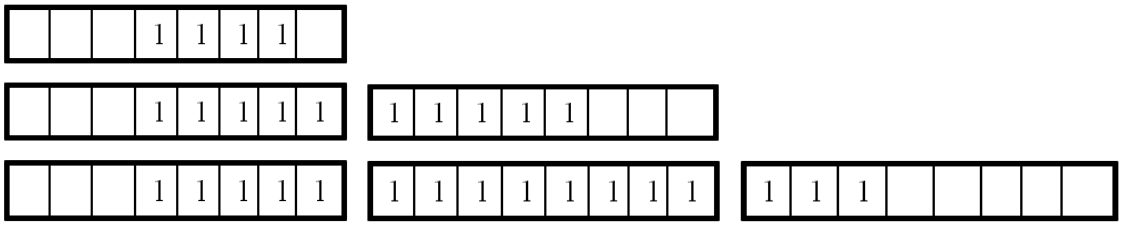


Рис.4.2. Варианты расположения горизонтального отрезка на растре.

В первом случае отрезок целиком помещается внутри байта (S1=S2). Во втором – занимает два байта(S1+1=S2), в третьем – три байта и более (S2>S1+1). Следовательно, программа должна вначале идентифицировать, с каким вариантом имеет дело. Рассмотрим наиболее сложный, третий случай.

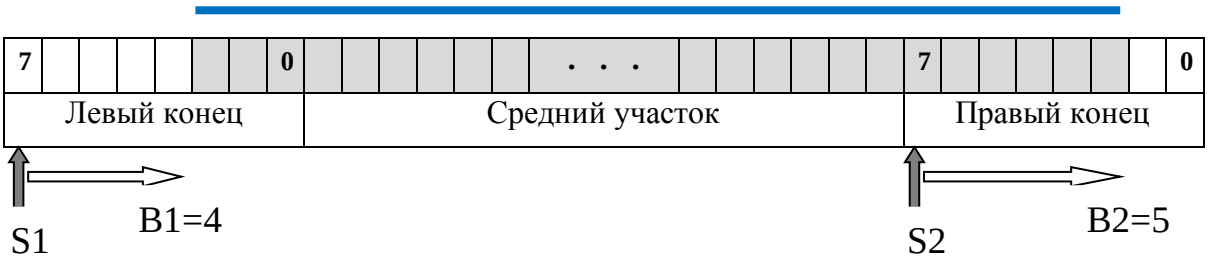


Рис. 4.3. Горизонтальный отрезок на битовом растре.

Как видно из рисунка 4.3, границы отрезка располагаются внутри байтов, следовательно, концы отрезков должны отображаться иначе, чем средний участок. Для начала и конца отрезка здесь должны быть вычислены смещения байтовые S1, S2 и битовые B1и B2

Средний участок может рисоваться тем же кодом, который приведен

выше для байтовой организации памяти, увеличив S1 на единицу. Рисование концов отрезка требует установки битовой маски таким образом, чтобы все бит начала отрезка и все биты правее его были установлены в единицу. Проще всего это сделать с помощью таблицы, как показано ниже.

```
B1    DB    ?             ; битовое смещ-е левого конца отрезка
TL    DB    11111111b     ; таблица
      DB    01111111b     ; для
      DB    00111111b     ; левого
      DB    00011111b     ; конца
      DB    00001111b     ; отрезка
      DB    00000111b     ;
      DB    00000011b     ;
      DB    00000001b     ;
      …
      XOR   DH, DH        ; очищаем старший байт регистра DX
      MOV   DL, B1        ; загружаем битовое смещение
      MOV   SI, DX        ; загружаем индексный регистр
      MOV   AL, T+[SI]    ; загружаем маску в AL
```

Другие способы, например, с помощью сдвигов, займут большее число команд и, следовательно, памяти и времени. Представленный выше код только в исходном тексте занимает много места. На самом деле таблица имеет размер всего восемь байт, а код фрагмента программы состоит из четырех команд. Аналогично можно организовать отображение правого конца отрезка.

4.2. Построение отрезка произвольной прямой на растре

При построении горизонтальных и вертикальных прямых на растре мы не рассматривали проблему округления, неизбежно возникающую при переносе отображаемого объекта на дискретный растр. Горизонтальные и вертикальные линии отображаются на растре идеально, погрешности округления могут быть заметными только в местах соединения отрезков. В случае построения наклонных отрезков мы сталкиваемся с проблемой отображения линии на растр.

Пусть начало и конец отрезка заданы координатами *x1*,*y1* и *x2*,*y2*. Эти координаты указывают на конкретные пиксели на экране (рис. 4.4). При этом остальные точки отрезка не находятся точно в центре пикселей.

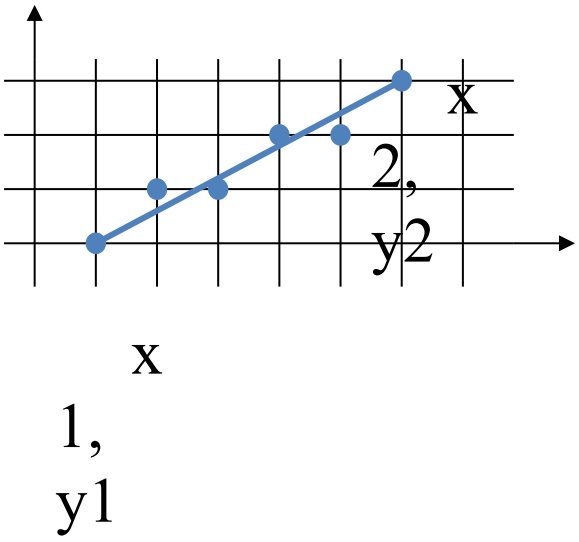


Рис. 4.4. Отображение наклонного отрезка на растр.

Следовательно, необходимо выбирать пиксели, ближайшие к идеальной траектории. Следует заметить, что в любом случае при рассмотрении экрана с увеличением зубцы будут заметными. Эта проблема актуальна не только для вывода графики на экран, но также и при печати. Современные лазерные и струйные принтеры используют технологии устранения зубцов путем уменьшения диаметра точки или капли чернил для точек, сильно выбивающихся из ряда. Мы не рассматриваем здесь эти технологии.

Итак, для отображения наклонного отрезка прямой мы должны для всех x, $x1<x<x2$ вычислить координаты y всех точек с округлением по правилам округления на основании уравнения прямой

$$y=ax+b,$$

где a – тангенс угла наклона прямой к оси X, $a=(y2-y1)/(x2-x1)$, b – ордината точки пересечения прямой с осью Y. В случае, если $a>1$, то число вертикальных линий растра, на которых мы должны разместить точки, меньше числа горизонтальных линий (рис.4.5).

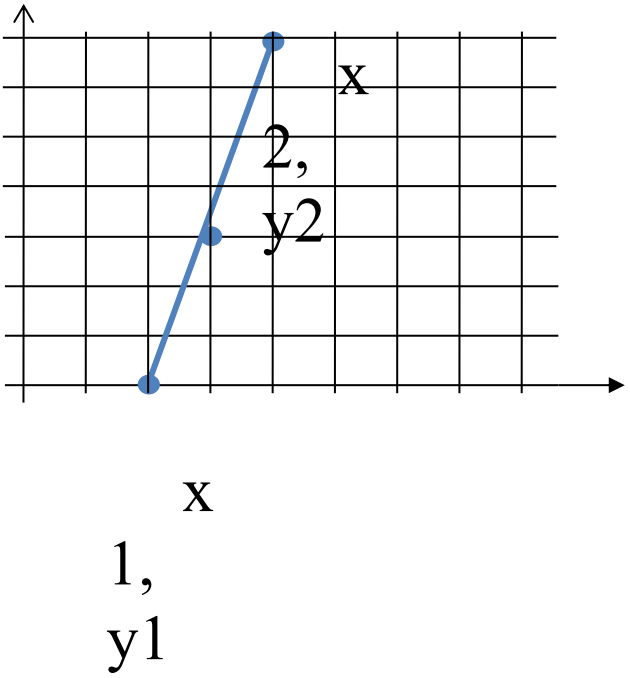


Рис. 4.5. Точки на растре при a>1

Решение, которое может устранить эту проблему, заключается в том, что для a>1 нужно решать обратную задачу, т.е. находить *x* для каждой точки *y, y1<y<y2* по формуле

$$x=y/a-b.$$

Заметим, что при *a=1* отрезок ложится на растр идеально. Далее будем рассматривать ситуацию *0<a<1*.

Вычисление координат точек по формуле $y=ax+b$ на первый взгляд не вызывает вопросов. Однако, операции умножения должны выполняться с плавающей точкой, либо с дробной частью. Поскольку отображаемый объект может состоять из многих отрезков, время его рисования с использованием умножений может существенно возрасти.

Избежать умножения можно, используя свойство дискретности растра. Приращения по *X* и *Y* равны $dx=1$, $dy=(y2-y1)/(x2-x1)$. Тогда

$$y_{i+1} = y_i+dy,$$

а округленное значение $round(y_{i+1})$ – это вертикальная координата на растре. Недостаток данного подхода заключается в том, что величина *dy* должна быть определена с высокой точностью. Предположим, что $dy = 1/3$ и точность его представления составляет один десятичный знак после запятой.

Табл.4.1. Накопление ошибки точности представления данных

	Приближенное	**Точное**

x	*y*	*round(y)*	*y*	*round(y)*
0	0	0	0	0
1	0.3	0	1/3	0
2	0.6	1	2/3	1
3	0.9	1	1	1
4	1.2	1	4/3	1
5	1.5	2	5/3	2
6	1.8	2	2	2
7	2.1	2	7/3	2
8	2.4	2	8/3	3

Как видим, уже на девятой итерации набегает ошибка в один пиксель. Величина погрешности зависит от угла наклона прямой, но можно считать, что число разрядов до запятой и после запятой должны быть одинаковыми. В этом случае набегающая погрешность не превысит одного пикселя.

Решить задачу построения отрезка прямой на растре в целочисленной арифметике позволяет итеративный **алгоритм Брезенхема**, изобретенный в 1965 г. (3) заключающийся в следующем. Для отрезка прямой вычисляются $dx = |(x2-x1)|$ и $dy = |(y2-y1)|$. Для каждой i-й точки отрезка вычисляется значение d_i, пропорциональное разности между s и t. При этом $d_0 = 2dy-dx$. Пусть пиксель P_{i-1} уже найден, и требуется выбрать пиксель T_i или S_i. Если $t > s$, то должен быть выбран пиксель S_i, иначе – T_i.

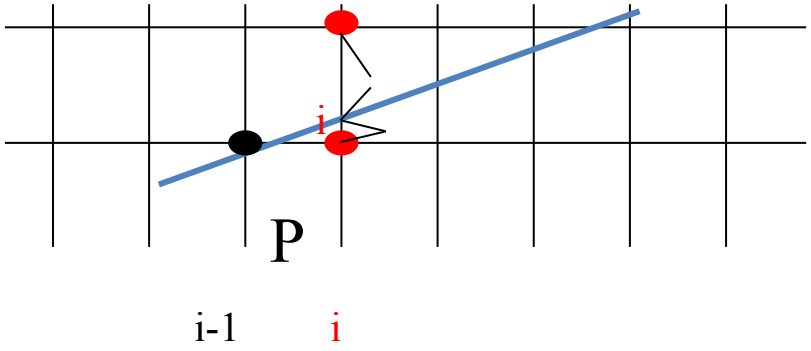


Рис.4.6. К алгоритму Брезенхема.

Тогда,

если $d_{i-1} \geq 0$, то $y_i = y_{i-1} + 1$, $d_i = d_{i-1} + 2(dy-dx)$,

иначе $y_i = y_i$, $d_i = d_{i-1} + 2dy$.

Таким образом, вычисление ординаты каждой следующей точки сводится к сложениям и вычитаниям, а умножение на два можно реализовать сдвигом на один разряд влево.

Рассмотрим, как алгоритм Брезенхема справится с предыдущим примером с соотношением $dy/dx = 1/3$. Пусть $dx = 9$, $dy = 3$, $d_0 = 6 - 9 = -3$.

i	x_i	d_i	y_i
0	0	-3	0
1	1	3	0
2	2	-9	1
3	3	-3	1
4	4	3	1
5	5	-9	2
6	6	-3	2
7	7	3	2
8	8	-9	3
9	9	-3	3

Рис. 4.7. Отрезок прямой, построенный по алгоритму Брезенхема.

Как видим, алгоритм Брезенхема позволяет вычислять координаты пикселей, ближайшие к идеальным позициям.

4.3. Построение окружности на растре

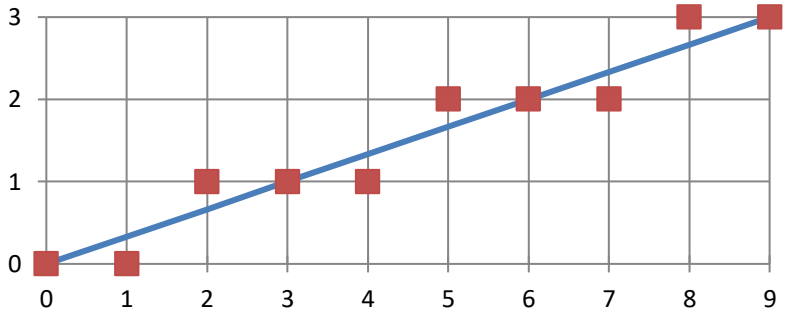
Как известно, уравнение окружности имеет вид:

$$x^2 + y^2 = r^2.$$

Пусть центр окружности находится в начале координат. В этом случае x и y будут совпадать с идеальными координатами окружности радиуса r, а находить координату y для каждого x мы можем по формуле

$$y = \pm\sqrt{r^2 + x^2}\ .$$

Правда, тогда на растре, где нет отрицательных координат, мы сможем отобразить только четверть окружности.



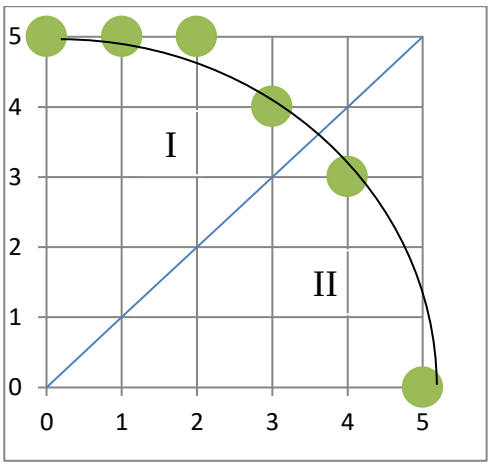


Рис. 4.8. Дуга окружности.

Так же, как при рисовании прямой, имеющей тангенс наклона к горизонтали больше единицы, при отображении дуги нам будет не хватать точек в секторе II (рис.4.8), следовательно, для построения окружности, следует использовать симметрию и составлять окружность из восьми дуг для сектора I.

Построение дуги как функции *y* on *x* представляет собой еще более сложную вычислительную задачу, поскольку включает не только умножения, но и извлечение квадратного корня. Существует итеративный алгоритм Брезенхема для построения окружности, позволяющий устранить эти проблемы.

Задача построения окружности на растре заключается в нахождении точек, ближайших к истинной окружности, т.е. в минимизации ошибки

$$e_i = (x_i^2 + y_i^2) - r^2$$

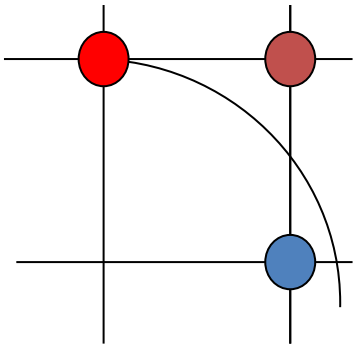


Рис. 4.9. Алгоритм Брезенхема для окружности

S_i

Как и в алгоритме для построения прямой, здесь используется управляющая переменная *d*, начальное значение которой $d_0 = 3 - 2r$. Для каждой *i*-й точки проверяется знак d_i.

T_i

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

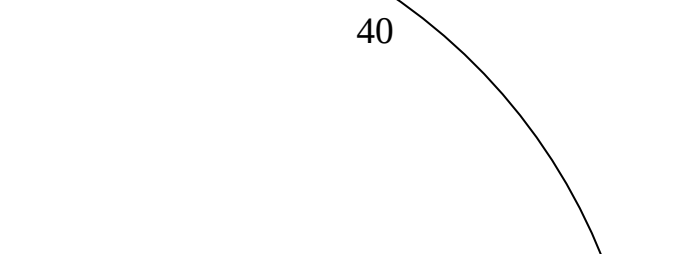
Если $d_i \geq 0$, то выбирается точка T_i и $d_{i+1} = d_i + 4(x_{i-1} - y_{i-1}) + 10$.

На каждый шаг итерации приходится две или три операции сложения,

одно сравнение и одно умножение на 4, которое может быть реализовано сдвигом влево на два разряда. Ниже приведен фрагмент кода, реализующего одну итерацию построения окружности с помощью алгоритма Брезенхема, из которого можно видеть, что в зависимости от того, по какой ветви алгоритма проходит управление, одна итерация занимает 8 или 11 команд процессора, причем в целых числах и без использования умножения. Максимальная ошибка не превышает ½ пикселя.

```
X       DW   ?       ; координата X
Y       DW   ?       ; координата Y
…
;  в регистре DX хранится значение переменной d
        CMP  DX, 0          ; сравниваем d с нулем
        JL   S              ; если меньше нуля, то это точка S
T:      MOV  AX, X          ; Точка T. Вычисляем новое d
        SUB  AX, Y          ; как X-Y
        SAL  AX, 1          ; умноженное
        SAL  AX, 1          ; на 4
        ADD  AX, 10         ; плюс 10
        ADD  DX, AX         ; плюс старое значение d
        DEC  Y              ; Уменьшаем Y
        JMP  Dspl
S:      MOV  AX, X          ; Точка S. Вычисляем новое D как X
        SAL  AX, 1          ; умноженное
        SAL  AX, 1          ; на 4
        ADD  AX, 6          ; плюс 6
        ADD  DX, AX         ; плюс старое значение d
Dspl:   INC  X              ; наращиваем X
…
```

	0	1	2	3	4	5	6	7	8	9
9	-17	-11	-1							
8				13	-1					



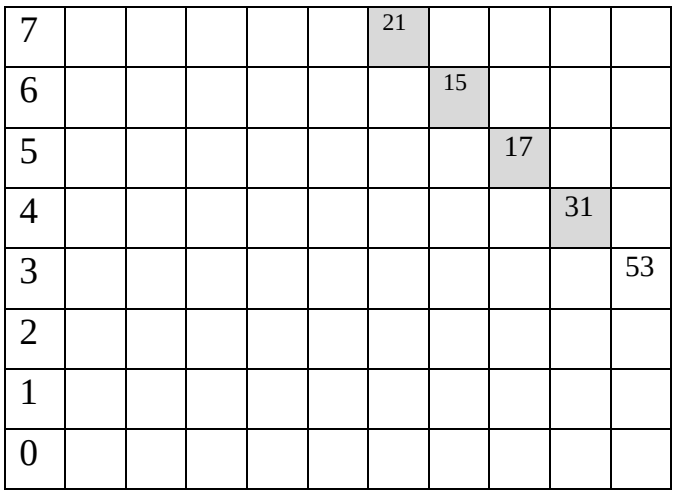


Рис. 4.10. Иллюстрация алгоритма Брезенхема.

Рис. 4.10 иллюстрирует изменение управляющей переменной d при построении дуги окружности радиуса 10 пикселей. Здесь также можно видеть, что этот алгоритм не работает при X>Y, поскольку должен построить одну точку при каждом увеличении X.

Ниже приведен еще один алгоритм построения окружности, точнее, дуги окружности (сектор I на рис.4.8). Автор этого алгоритма – Э.В. Стародубцев. **Алгоритм Стародубцева** заключается в следующем.

1. Координаты начальной точки – самой верхней точки окружности $x = 0, y = r$. Управляющая переменная $d = 0$.
2. Наращиваем x и вычисляем $d = d+i$.
3. Сравниваем d и y.
4. Если $d < y$, то координата y не меняется.
5. Если $d \geq y$ то $y = y - 1$, и $d = d - y$.
6. Выводим точку на экран и повторяем с п.2, пока $x < y$.

Ниже приведен фрагмент кода, реализующий одну итерацию вычисления координаты точки для дуги окружности с помощью данного алгоритма. Можно видеть, что этот код гораздо компактнее такого же кода для алгоритма Брезенхема. Алгоритм дает ошибку меньше, чем ½ пикселя в подавляющем большинстве случаев, и только для некоторых значений небольших радиусов (<10), ошибка может превышать ½ пикселя.

```
X       DW   ?              ; координата X
Y       DW   ?              ; координата Y
…
;  в регистре DX хранится значение переменной d (вначале d=0)
        INC   X             ; наращиваем X
        ADD  DX, X          ; увеличиваем d на величину X
        CMP  DX, Y          ; сравниваем с Y
        JL     Dspl         ; если меньше, то вывод точки
        SUB  DX, Y          ; иначе уменьшаем  d на Y
        DEC  Y              ; и уменьшаем Y на 1
Dspl: …
```

На рис. 4.11 показано, как меняется управляющая переменная при построении дуги радиуса 10 пикселей. В частности, темным цветом выделена точка (X=3, Y=9), отстоящая от идеальной дуги более, чем на ½ пикселя.

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

Рис. 4.11. Иллюстрация алгоритма Стародубцева

Для устранения ошибок при построении окружностей малых радиусов алгоритм можно модифицировать: на каждом шаге итерации сравнивать значение управляющей переменной *d* не с *y*, а с *y/2*. Деление на два можно реализовать сдвигом вправо.

	0	1	2	3	4	5	6	7	8	9

9	0	1	3	6						
8				-2	2	7				
7						0	6			
6							0	7		
5								2	10	
4									6	
3									3	
2										
1										
0										

Рис. 4.12. Иллюстрация улучшенного алгоритма Стародубцева.

Если требуется построить целую окружность или дугу, занимающую более одного квадранта, то целесообразно на каждом шаге итерации сразу строить восемь точек (для полной окружности), используя свойство симметрии.

4.4. Устранение лестничного эффекта

Несмотря на то, что точки на наклонных или кривых линиях располагаются оптимальным образом, на них можно увидеть зубцы или лесенки. Особенно сильно этот эффект обнаруживается, когда угол наклона линии близок к нулю или к 90°. При растровом синтезе изображения положение пикселей изменить невозможно, но есть возможность сгладить зубцы путем изменения яркости, как показано на рис. 4.13.

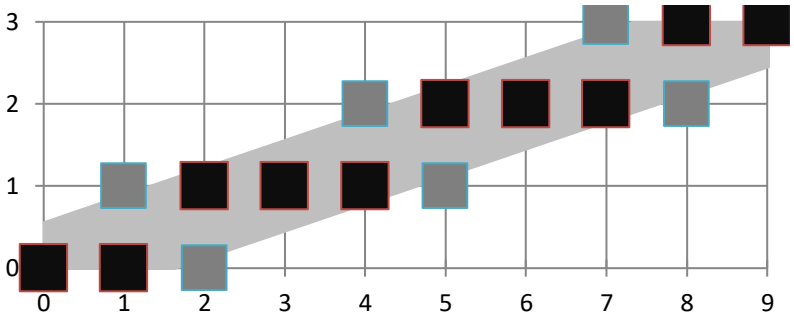


Рис. 4.13. Сглаживание методом размывания краев.

Чем дальше центр пикселя от границы линии, тем меньше интенсивность его цвета. Такой метод хорошо работает только на толстых линиях, иначе тонкая линия превратится в пунктир. Другая проблема – последующая закраска (материал следующей главы) может дать заметную границу между контуром и закрашиваемым полем.

ГЛАВА 5. АЛГОРИТМЫ ЗАКРАСКИ ОБЛАСТЕЙ

5.1. Методы закраски областей

В главе 4 мы рассматривали построение линий на растре, с помощью которых можно изобразить такие геометрические фигуры, как окружность, квадрат, треугольник и др. Одна и та же фигура может быть отображена на растре как набор отрезков (каркас), и как часть плоскости, ограниченная этими отрезками (рис.5.1).

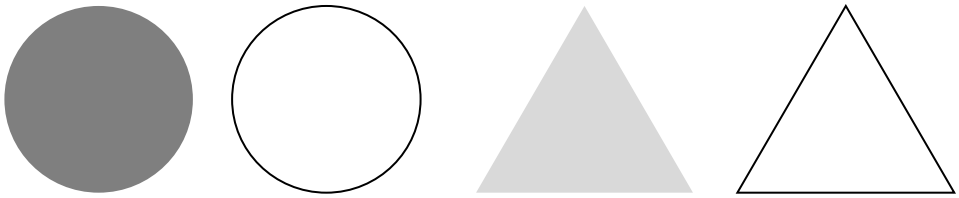


Рис. 5.1. Фигуры как наборы отрезков и как части плоскости

Так, математическое определение круга – это геометрическое место точек, расстояние которых от центра не превышает величину радиуса. Другое определение круга – часть плоскости, ограниченная окружностью. Таким образом, почти каждая плоская фигура может быть определена двумя способами. Из этого вытекают два способа построения закрашенных фигур.

Первый способ – **растровая развертка**, которая заключается в том, что заполнение фигуры точками делается на основании математической модели объекта. Необходимым условием применимости данного метода является наличие такой математической модели. Очевидно, что рассмотренные в предыдущей главе итеративные методы как раз и позволяют строить линии в обход математических моделей.

Второй способ – **затравочное заполнение** – предполагает предварительное построение каркасной фигуры, состоящей из линий, после чего точки внутри ограниченной области закрашиваются заданным цветом. Закраска начинается с некоторой точки внутри фигуры – **затравочной точки**, которая закрашивается требуемым цветом, после чего цвет этой точки распространяется на соседние до тех пор, пока не будут закрашены все точки внутри области. Каждая закрашенная точка становится новой затравочной точкой. Этот способ получил название из-за сходства с процессом кристаллизации и пригоден для закраски как фигур, построенные с помощью линий, так и для изображений, которые изначально были растровыми, например, цифровых фотографий.

5.2. Заполнение многоугольников

Простейший способ закраски многоугольника состоит в том, чтобы пройти по всему растру, проверяя принадлежность каждой точки к закрашиваемой фигуре. Данный метод является расточительным, поскольку число таких проверок для любой фигуры будет равно количеству точек на растре.

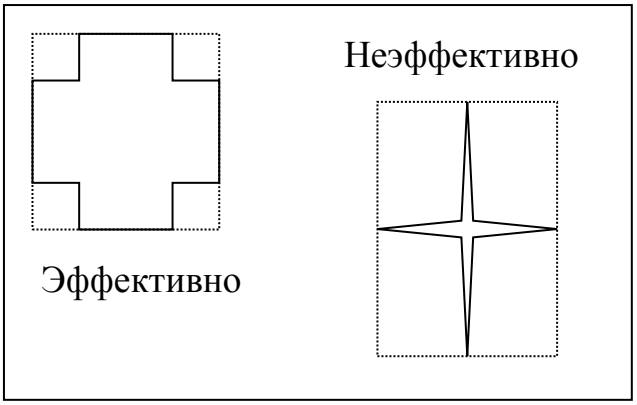


Рис.5.2. Выделение прямоугольной оболочки на экране

Можно ограничить область растра, в которой делаются такие проверки, прямоугольником, в который вписана фигура. В одних случаях, например, когда многоугольник выпуклый, это эффективно, в других случаях – недостаточно эффективно (рис. 5.2).

Видеопамять организована таким образом, чтобы максимально упростить вывод изображения на ЭЛТ посредством строчной и кадровой развертки, из чего следует, что закраску областей на экране также лучше вести по строкам, причем каждую строку желательно обрабатывать независимо от соседних строк. Иными словами, задачу закраски двумерной фигуры нужно сделать линейной.

Если рассматривать отдельно взятую строку, то программе она представляется цепочками точек с одинаковыми атрибутами, которые прерываются точками, через которые проходит контур фигуры. Такое свойство изображения называется пространственной когерентностью. Строго говоря, для выявления границ фигуры достаточно, чтобы атрибуты точек, составляющих контур фигуры, были идентифицируемыми, и не встречались внутри или снаружи фигуры. На рис. 5.3 показано, что сканирующая строка 2 делится на три области: область $x<1$ лежит вне многоугольника, $1 \le x \le 8$ находится внутри многоугольника, $x>8$ – вне многоугольника. Сканирующая строка 4 делится на пять областей:

$x<1$ – вне многоугольника;

$1 \le x \le 4$ – внутри МУ;

$4 < x < 6$ – вне МУ;

$6 \le x \le 8$ – внутри МУ;

$x > 8$ – вне МУ

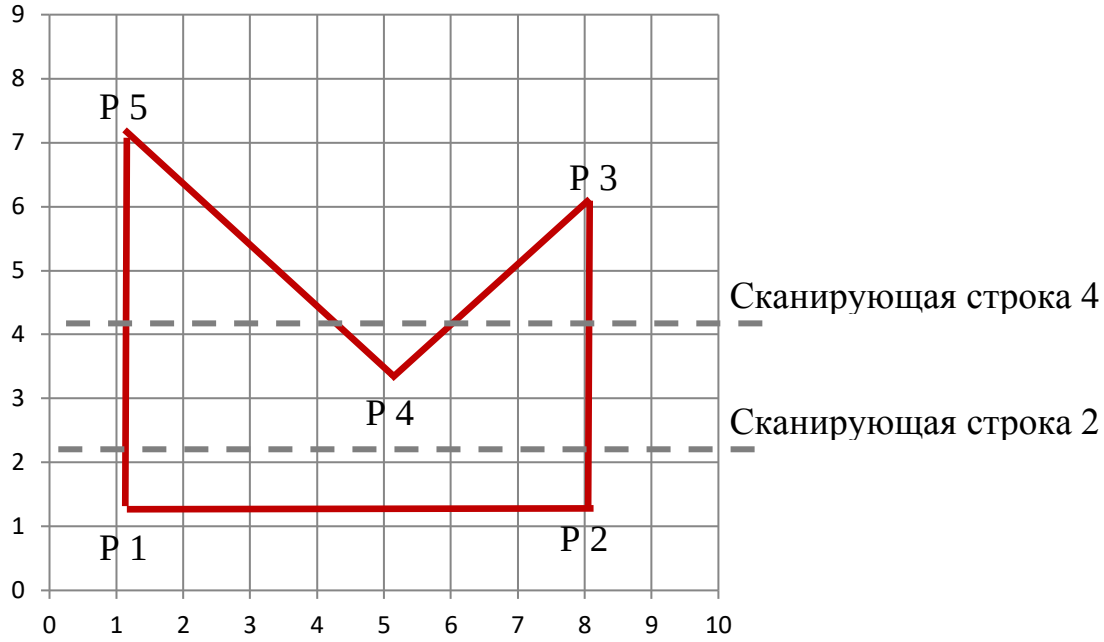


Рис. 5.3. Пространственная когерентность.

Предположим, что для сканирующей линии точки пересечения этой лини с ребрами многоугольника известны. Для сканирующей строки 4 это будут точки со смещениями 1, 4, 6, 8. Второе предположение: многоугольник целиком лежит в пределах растра. В таком случае закраска строки сводится к следующим шагам:

1. Отсортировать смещения точек по возрастанию (если необходимо).
2. Закрашивать цветом фона точки до первой точки пересечения с ребром.
3. Закрашивать цветом многоугольника до следующего пересечения.
4. Повторить с п.2 до конца строки.

При этом горизонтальные ребра нужно игнорировать, т.е. последовательность точек цвета ребра многоугольника считать одной точкой.

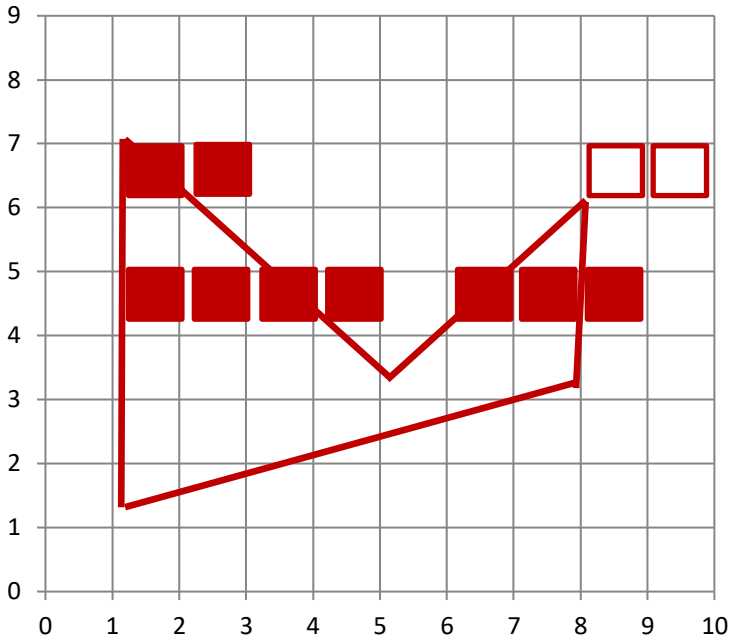


Рис.5.4. Закраска строки.

Игнорировать также следовало бы и углы многоугольника. На рис. 5.4 видно, что закраска строки 6 приведет к сбою рисунка. Однако в этом случае незакрашенными останутся все точки строки 1 на рис. 5.4. Кроме того, что существует проблема, связанная со строгостью сравнения. Сканирующая строка 4 пересекает многоугольник в точках 1, 4, 6, 8. Если закрашивать пиксели цветом заливки многоугольника по строгому сравнению, то будут закрашены пиксели 2, 3, 7, а по нестрогому сравнению – 1, 2, 3, 4, 6, 7, 8, как показано на рис. 5.4. В первом случае незакрашенные пиксели будут внутри многоугольника, а во втором – снаружи. Можно устанавливать для нечетного пересечения нестрогое сравнение, а для четного – строгое, но это не устраняет проблему вершин многоугольника (сканирующие линии 7 и 8 на рис. 5.4).

Решение, которое позволит устранить все указанные недостатки, заключается в том, что границы многоугольника по-прежнему лежат на границе пикселей, но сканирующие строки проходят через середины пикселей. Сканирующая строка 4.5 будет пересекать многоугольник в точках 1, 3.5, 6.5, 8. Закрашивать следует пиксели, центры которых находятся справа от пересечения внутри интервала, т.е. 1, 2, 3, 7 (рис. 5.5).

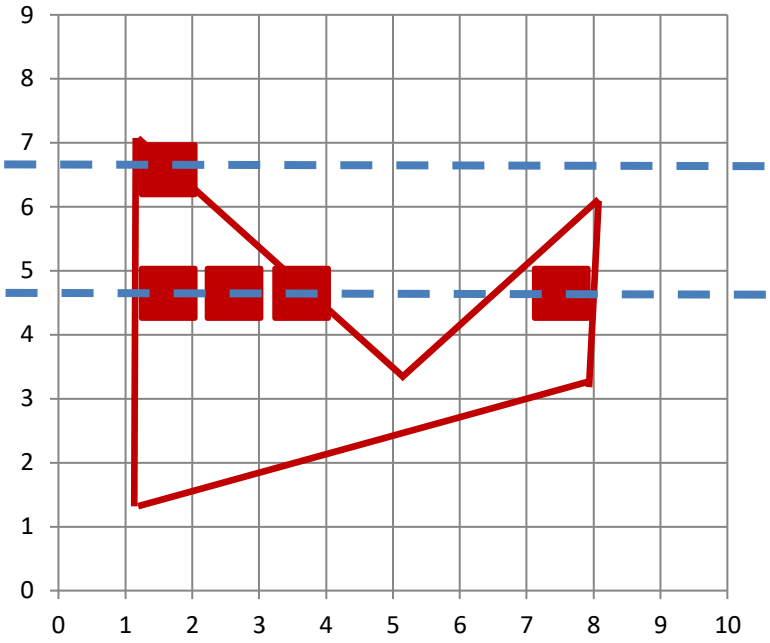


Рис.5.5. Сканирование через центры пикселей.

Сканирование через центры пикселей позволяет также автоматически игнорировать горизонтальные ребра многоугольников, поскольку они не имеют общих точек ни с одной сканирующей линией.

5.3. Алгоритм закраски с упорядоченным списком ребер

Прежде всего, необходимо найти все точки пересечения сканирующих линий с ребрами многоугольника (рис. 5.6).

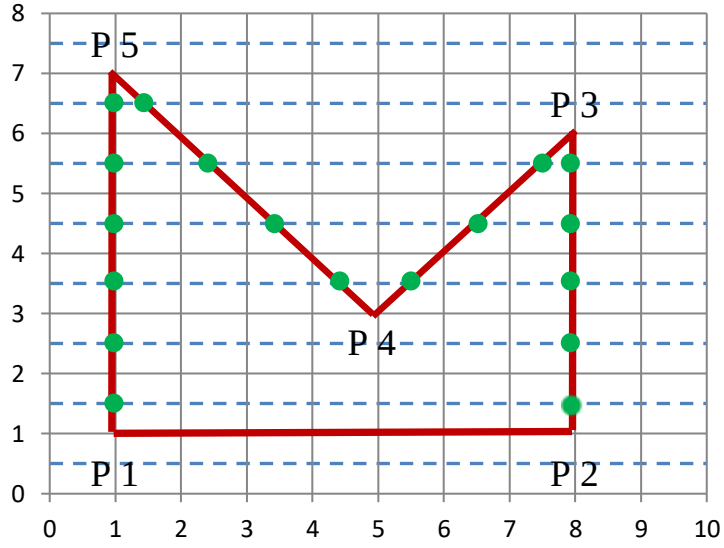


Рис.5.6. Точки пересечения сканирующих линий с ребрами многоугольника.

Чтобы найти эти точки, нужно решить системы уравнений:

Для ребра P2P3:

x=8

x=1,5 => (8;1.5) и т.д.

Для ребра P4P4:

y=x – 2

y = 5,5 => X = Y+2 = 7,5 => (7,5;5,5) и т.д.

В результате решения всех уравнений такими точками будут:

- для ребра P2P3: (8; 1,5), (8; 2,5), (8; 3,5), (8; 4,5), (8; 5,5);
- для ребра P3P4: (5,5; 3,5), (6,5; 4,5), (7,5; 5,5);
- для ребра P4P5: (4,5; 3,5), (3,5; 4,5), (2,5; 5,5), (1,5; 6,5);
- для ребра P5P1: (1; 1,5), (1; 2,5), (1; 3,5), (1; 4,5), (1; 5,5), (1; 6,5).

Полученный список сортируется в порядке сканирования (сверху вниз и слева направо):

(1; 6,5), (1,5; 6,5), (1; 5,5), (2,5; 5,5), (7,5; 5,5), (8; 5,5),

(1; 4,5), (3,5; 4,5), (6,5; 4,5), (8; 4,5), (1; 3,5), (4,5; 3,5),

(5,5; 3,5), (8; 3,5), (1; 2,5), (8; 2,5), (1; 1,5), (8; 1,5)

После этого выделяем на списке пары пересечений, лежащие на одной горизонтали:

(1; 6,5), (1,5; 6,5), (1; 5,5), (2,5; 5,5), (7,5; 5,5), (8; 5,5),

(1; 4,5), (3,5; 4,5), (6,5; 4,5), (8; 4,5), (1; 3,5), (4,5; 3,5),

(5,5; 3,5), (8; 3,5), (1; 2,5), (8; 2,5), (1; 1,5), (8; 1,5)

Таким образом, получается девять горизонтальных отрезков.
Применяя алгоритм описанный в подразд.5.2., закрашиваем пиксели:

(1, 6)

(1, 5), (2, 5), (7, 5)

(1, 4), (2, 4), (3, 4), (6, 4), (7, 4)

(1, 3), (2, 3), (3, 3), (4, 3), (5, 3), (6, 3), (7, 3)

(1, 2), (2, 2), (3, 2), (4, 2), (5, 2), (6, 2), (7, 2)

(1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1)

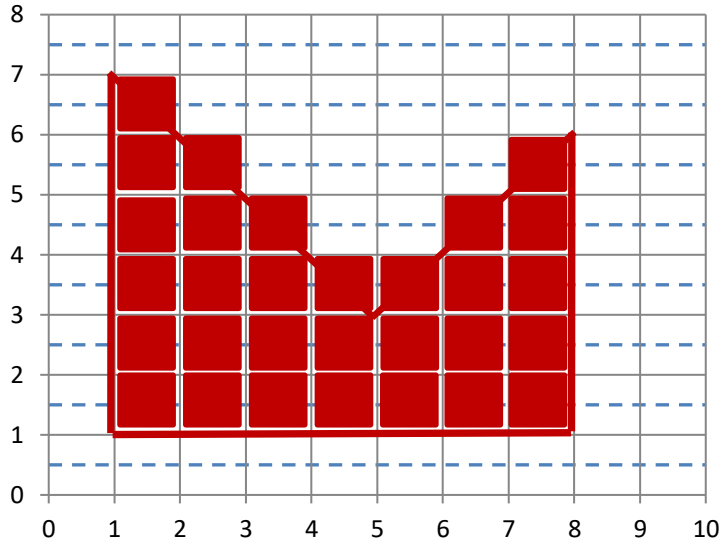


Рис. 5.7. Результат работы алгоритма закраски с упорядоченным списком ребер.

Недостаток алгоритма закраски с упорядоченным списком ребер состоит в том, что список точек пересечения сканирующих прямых с ребрами многоугольника может быть очень большим, что приводит к большим затратам времени на создание, поддержку и сортировку больших списков.

5.4. Алгоритм заполнения многоугольника по ребрам

Данный алгоритм не требует сортировки списков и может использовать данные о пересечении сканирующих прямых с ребрами многоугольника в произвольном порядке. Суть алгоритма заключается в том, что для каждой точки пересечения (x_i, y_i) выполняется перекраска всех пикселей, центры которых лежат справа от x_i, т.е. $x+1/2>x_i$. Термин перекраска означает, что цвет фона перекрашивается в цвет многоугольника, а цвет многоугольника – в цвет фона.

Рассмотрим алгоритм заполнения многоугольника по ребрам на примере четвертой строки рассмотренного выше многоугольника. Точки пересечения сканирующей строки 4,5 и ребер многоугольника следующие (этот список содержит точки сканирующей прямой в произвольном порядке):

(1; 4,5), (8; 4,5), (6,5; 4,5), (3,5; 4,5)

Выполним последовательно для этих точек операцию перекраски пикселей справа от каждой точки:

0	1	2	3	4	5	6	7	8	9	-- исходная строка
0	1	2	3	4	5	6	7	8	9	-- после обработки точки (1; 4,5)
0	1	2	3	4	5	6	7	8	9	-- после обработки точки (8; 4,5)
0	1	2	3	4	5	6	7	8	9	-- после обработки точки (6,5; 4,5)
0	1	2	3	4	5	6	7	8	9	-- после обработки точки (3,5; 4,5)

Данный алгоритм вообще не требует какой-либо упорядоченности обрабатываемых точек. Его недостатком является возможная многократная перекраска одних и тех же пикселей. Если считать, что при обработке каждой точки в среднем перекрашивается половина строки, то число перекрашиваемых пикселей n равно

$$n = h*L/2,$$

где h – высота многоугольника в пикселях, L – длина строки экрана в пикселях. Для закраски квадрата 10x10 пикселей на экране с длиной строки 1280 число перекрашиваемых пикселей составит 6400 вместо 100, т.е. в 64 раза больше!

5.5. Алгоритм заполнения по ребрам с перегородкой

Для уменьшения числа перекрашиваемых пикселей при заполнении многоугольника по ребрам можно ввести т.н. перегородку – вертикальную линию, которая обычно проводится через одну из вершин многоугольника (рис.5.8).

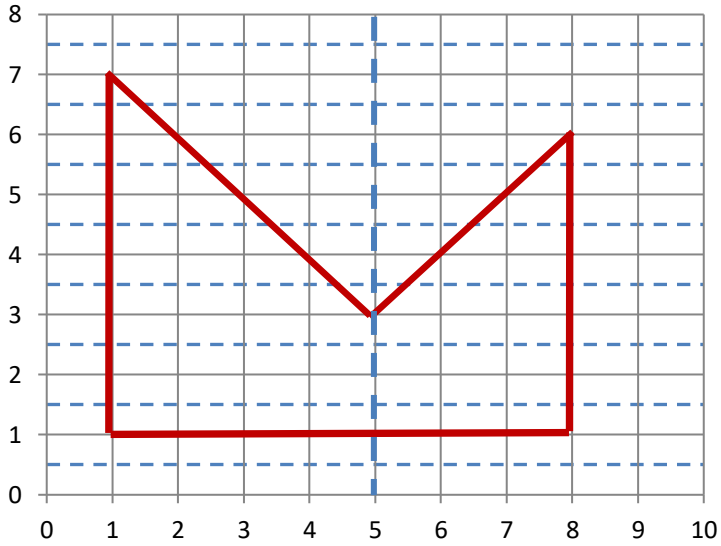


Рис.5.8. К определению заполнения по ребрам с перегородкой.

Алгоритм заполнения по ребрам с перегородкой состоит в том, что

выполняются два алгоритма заполнения многоугольника по ребрам навстречу перегородке: выполняется перекраска всех пикселей от каждой точки до перегородки.

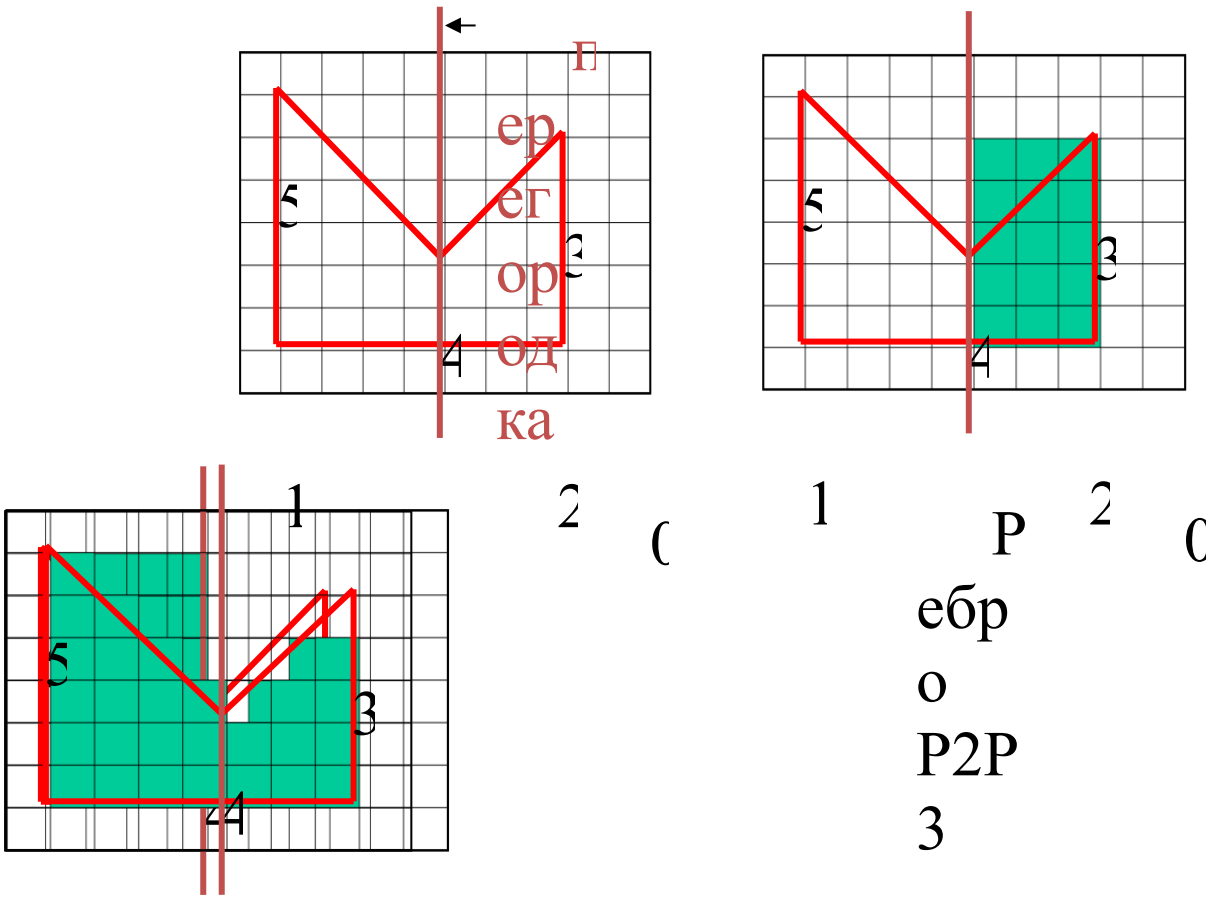


Рис.5.9. Работа алгоритма заполнения по ребрам с перегородкой.

5.6. Алгоритм со списком ребер и флагом

Данный алгоритм состоит из двух шагов. На первом шаге обрисовывается контур многоугольника путем окраски одного пикселя рядом с каждой точкой пересечения сканирующей прямой с ребром многоугольника. Окрашивается ближайший пиксель i, центр которого лежит справа от пересечения x, т. е. $x_i \geq ½ + x$. На рис. 5.10 показан результат обрисовки контура, из которого видно, что контур не совпадает с идеальным положением многоугольника, а сдвинут вправо на 0,5 – 1 пиксель.

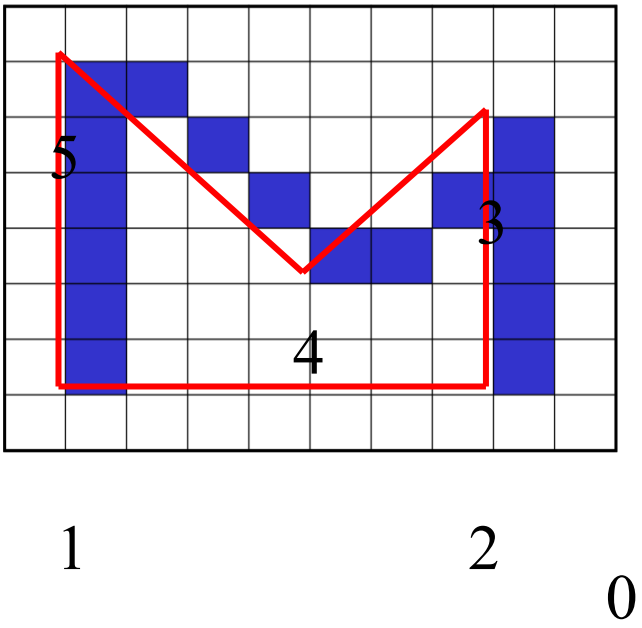


Рис.5.10. Результат обрисовки контура.

На втором шаге выполняется заполнение – закрашивание пикселей, расположенных между пикселями, ограничивающими контур. Закраска управляется переменной-триггером. В начале каждой строки триггер устанавливается в ноль, и для всех пикселей строки выполняются следующие действия:

1. Если пиксель имеет цвет контура, то состояние триггера инвертируется.
2. Если триггер установлен в 1, пиксель окрашивается цветом многоугольника, иначе – цветом фона.

Ниже показан процесс перекраски третьей строки для многоугольника с обрисованным контуром. Пусть многоугольник должен быть красным, а фон – серым. Обрисовка контура выполнена синим.

Исходное состояние:

0	1	2	3	4	5	6	7	8	9

x=0: триггер =0; цвет = серый

0	1	2	3	4	5	6	7	8	9

x=1: триггер =0–>1; цвет = красный

0	1	2	3	4	5	6	7	8	9

x=2: триггер =1; цвет = красный

0	1	2	3	4	5	6	7	8	9

x=3: триггер =1; цвет = красный

0	1	2	3	4	5	6	7	8	9

x=4: триггер =1; цвет = красный

0	1	2	3	4	5	6	7	8	9

x=5: триггер =1–>0; цвет = серый

0	1	2	3	4	5	6	7	8	9

x=6: триггер =0–>1; цвет = красный

0	1	2	3	4	5	6	7	8	9

x=7: триггер =1; цвет = красный

0	1	2	3	4	5	6	7	8	9

x=8: триггер =1–>0; цвет = серый

0	1	2	3	4	5	6	7	8	9

x=9: триггер =0; цвет = серый

0	1	2	3	4	5	6	7	8	9

Таким образом, строка закрашена за один раз.

5.7. Простой алгоритм закраски с затравкой

Алгоритмы закраски фигур с затравкой предусматривают распространение закраски пикселей во все стороны от произвольно выбранной затравочной точки к границам заполняемой фигуры.

В рассмотренном выше методе закраски с предварительно обрисованным контуром контур обрисовывался справа, а в алгоритме было предусмотрено переключение триггера на точках, принадлежащих контуру, Тем самым смещение контура вправо устранялось, поскольку раскраска пикселя происходила уже после переключения триггера. В случае распространения закраски во все стороны от затравочной точки мы не можем себе позволить сдвиг обрисованного контура в сторону. Нам остаются две возможности ограничения закрашиваемой области: снаружи или внутри контура.

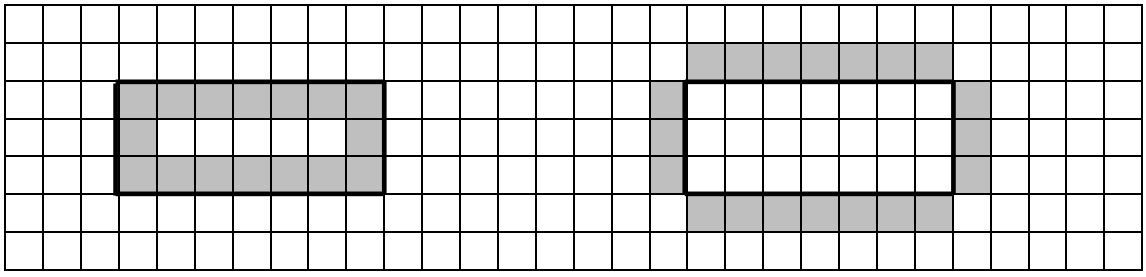


Рис. 5.11. Внутренне-определенная и гранично-определенная области.

В случае внутренне-определенной области (рис.5.11 слева) в процесс раскраски должны вовлекаться точки, ограничивающие контур. Для гранично-определенной области раскраска не затрагивает контур. Применение того или иного метода оконтуривания зависит от конкретной задачи. Например, если требуется сохранить каркас рисунка, то должны использоваться гранично-определенные области.

Простой алгоритм закраски с затравкой является рекурсивным. Каждый закрашенный пиксель становится затравочным, и процесс распространяется во все стороны. Рекурсивные алгоритмы являются простыми в реализации и компактными, но довольно расточительными. Данный алгоритм состоит из следующих шагов:

1. Затравочный пиксель помещается в стек.
2. Пиксель извлекается из стека.
3. Пиксель закрашивается.
4. Для каждого из соседних пикселей: если пиксель является закрашенным или граничным, пиксель игнорируется, иначе пиксель помещается в стек.
5. Повторить с п.2 пока стек не пуст.

Рассмотрим работу простого алгоритма закраски с затравкой. В качестве фигуры используем прямоугольник с размерами 6x3 пикселей с гранично-определенной областью. Для простоты все пиксели имеют сквозную нумерацию. Обход окружения совершается по часовой стрелке: →↓←↑. Пусть в качестве затравочного пикселя взят пиксель номер 10.

1. Затравочный пиксель 10 помещается в стек.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18

Стек

10						

2. Пиксель 10 извлекается из стека. Он закрашивается, а все соседние пиксели 11, 16, 9, 4 помещаются в стек.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18

Стек

4	9	16	11								

3. Пиксель 4 извлекается из стека и закрашивается, а пиксели 5, 3 помещаются в стек.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18

Стек

3	5	4	9	16	11						

4. Пиксель 3 извлекается из стека, закрашивается, а пиксели 9, 2

помещаются в стек.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18

Стек

2	9	5	4	9	16	11					

5. Пиксель 2 извлекается из стека, закрашивается, а пиксели 8, 1 помещаются в стек.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18

Стек

1	8	9	5	4	9	16	11				

6. Пиксель 1 извлекается из стека, закрашивается, а пиксель 7 помещается в стек.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18

Стек

7	8	9	5	4	9	16	11				

7. Пиксель 7 извлекается из стека, закрашивается, а пиксели 8, 13 помещаются в стек.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18

Стек

13	8	8	9	5	4	9	16	11			

8. Пиксель 13 извлекается из стека и закрашивается, а пиксель 14 помещается в стек.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18

Стек

14	8	8	9	5	4	9	16	11			

9. Пиксель 14 извлекается из стека и закрашивается, а пиксели 15, 8 помещаются в стек.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18

Стек

8	15	8	8	9	5	4	9	16	11		

10. Пиксель 8 извлекается из стека и закрашивается, а пиксель 9 помещается в стек.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18

Стек

9	15	8	8	9	5	4	9	16	11		

11. Пиксель 9 извлекается из стека и закрашивается, а пиксель 15 помещается в стек.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18

Стек

15	15	8	8	9	5	4	9	16	11		

12. Пиксель 15 извлекается из стека и закрашивается, а пиксель 16 помещается в стек.

1	2	3	4	5	6

Стек

7	8	9	10	11	12
13	14	15	16	17	18

16	15	15	8	8	9	5	4	9	16	11	

13. Пиксель 16 извлекается из стека и закрашивается, а пиксель 17 помещается в стек.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18

Ст

17	15	15	8	8	9	5	4	9	16	11	

14. Пиксель 17 извлекается из стека и закрашивается, а пиксели 18 и 11 помещаются в стек.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18

Стек

11	18	15	15	8	8	9	5	4	9	16	11

15. Пиксель 11 извлекается из стека и закрашивается, а пиксели 12 и 5 помещаются в стек.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18

Стек

5	12	18	15	15	8	8	9	5	4	9	16	11

16. Пиксель 5 извлекается из стека и закрашивается, а пиксель 6 помещается в стек.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18

Стек

6	12	18	15	15	8	8	9	5	4	9	16	11

17. Пиксель 6 извлекается из стека и закрашивается, а пиксель 12 помещается в стек.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18

Стек

12	12	18	15	15	8	8	9	5	4	9	16	11

17. Пиксель 12 извлекается из стека и закрашивается, а пиксель 18 помещается в стек.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18

Стек

18	12	18	15	15	8	8	9	5	4	9	16	11

18. Пиксель 18 извлекается из стека и закрашивается, а в стек поместить нечего.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18

Стек

18	12	18	15	15	8	8	9	5	4	9	16	11

Дальнейшие действия заключаются в том, что извлекаемые из стека пиксели закрашиваются повторно, и делается проверка соседних пикселей, которая, естественно, не добавляет новых пикселей в стек.

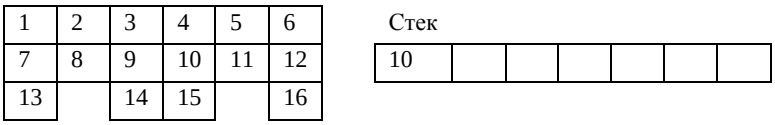
Возможная модификация данного алгоритма заключается в том, что перед закрашиванием цвет пикселя проверяется, и если он уже закрашен,

то проверки соседних пикселей не производятся. В таком случае завершение закраски будет состоять в проверке цвета извлекаемых пикселей. Стековая организация памяти не предоставляет возможности быстрой проверки наличия числа в памяти, поэтому повторное занесение пикселей в стек неизбежно.

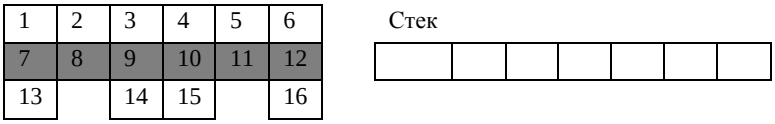
5.8. Построчный алгоритм закраски с затравкой

1. Затравочный пиксель помещается в стек.
2. Пиксель извлекается из стека.
3. Строка с пикселем закрашивается влево и вправо до границ.
4. Найденные границы запоминаются в переменных $X_{лев}$ и $X_{прав}$.
5. В диапазоне $X_{лев} \le x \le X_{прав}$ проверяются строки, расположенные непосредственно над и под текущей строкой. Если в них есть незакрашенные пиксели, то для этих двух строк находится крайний правый незакрашенный пиксель в каждом интервале и помещается в стек.
6. Повтор с р.2 пока стек не пуст.

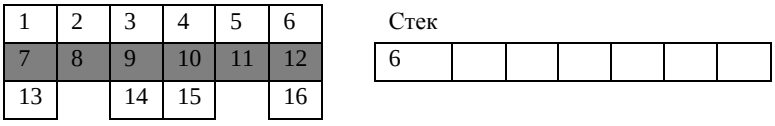
1. Затравочный пиксель 10 помещается в стек.



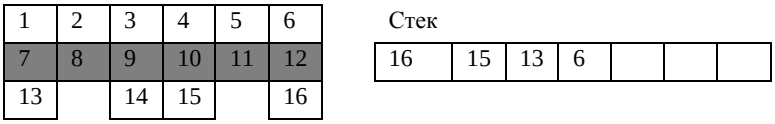
2. Пиксель извлекается из стека. Строка с пикселем закрашивается влево и вправо до границ.



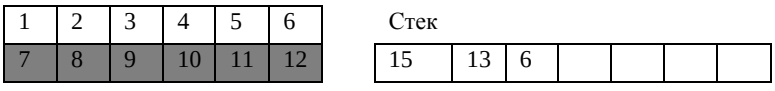
3. Проверяется строка сверху. Самый правый пиксель помещается в стек.



4. Проверяется строка снизу. В стек помещаются пиксели 13, 15, 16.



5. Пиксель извлекается из стека. Окрашивается пиксель 16.



13		14	15		16

6. Проверяется строка выше текущей. Неокрашенных пикселей нет.

7. Пиксель извлекается из стека. Окрашиваются пиксели 15, 14.

1	2	3	4	5	6
7	8	9	10	11	12
13		14	15		16

Стек

13	6					

8. Проверяется строка выше текущей. Неокрашенных пикселей нет.

9. Пиксель извлекается из стека. Окрашивается пиксель 13.

1	2	3	4	5	6
7	8	9	10	11	12
13		14	15		16

Стек

6						

10. Проверяется строка выше текущей. Неокрашенных пикселей нет.

11. Пиксель извлекается из стека. Окрашивается верхняя строка.

1	2	3	4	5	6
7	8	9	10	11	12
13		14	15		16

Стек

12. Проверяется строка ниже текущей. Неокрашенных пикселей нет.

13. Стек пуск. Работа алгоритма завершена.

ГЛАВА 6. ОТСЕЧЕНИЕ

6.1. Назначение отсечения

Изображение, построенное на экране, не всегда состоит из объектов, целиком помещающихся на экран. Часто картинка на экране – это фрагмент большой картины, например, в случае ее увеличения. В таких случаях необходимо отсекать часть фигур, делая их невидимыми. Кроме этого отсечение части объектов необходимо для удаления невидимых линий в трехмерных объектах, при формировании теней и в некоторых других случаях. Алгоритмы отсечения бывают двумерными и трехмерными. Принципы отсечения идентичны методам начертательной геометрии, применяемым для нахождения пересечения простых тел (призмы, пирамиды) или поверхностей второго рода (цилиндры, конусы).

6.2. Отсечение прямоугольным окном

Пусть на экране имеется прямоугольное окно, стороны которого параллельны сторонам экрана. Координаты углов окна обозначим диагональными вершинами: левой нижней ($x_л$, $y_н$) и правой верхней ($x_п$, $y_в$). На экране произвольным образом расположены отрезки прямых. Необходимо отобразить эти отрезки при условии видимости только внутри прямоугольного окна.

Сначала рассмотрим самые простые случаи, когда отрезки находятся целиком внутри или целиком вне прямоугольного окна.

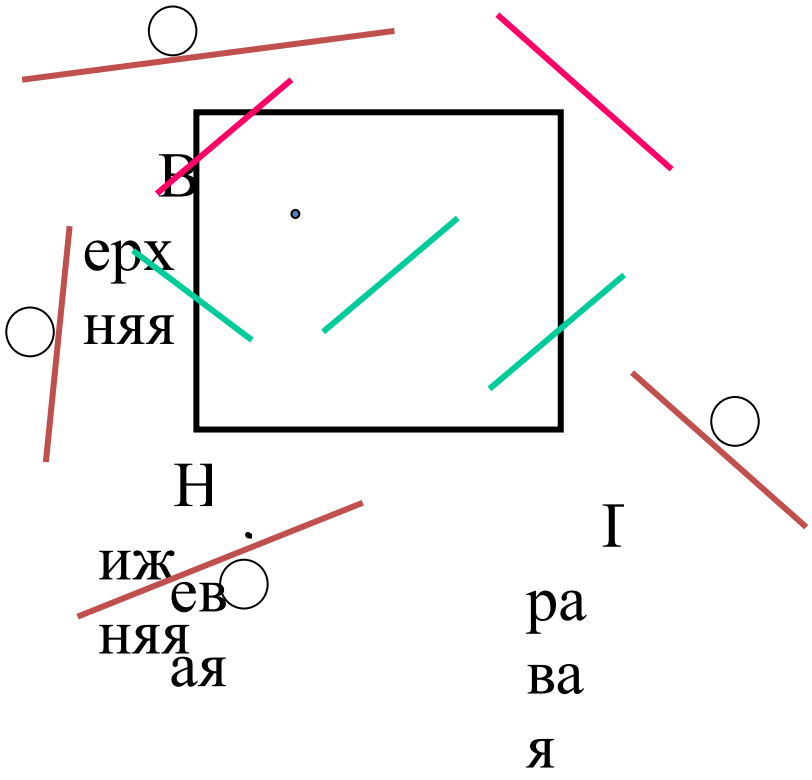


Рис.6.1. Прямоугольное отсекающее окно

Среди этих отрезков есть такие, которые находятся полностью выше, ниже, правее или левее окна (рис. 6.1). Такие отрезки обозначены цифрами 1, 2, 3, 4. Условие полной тривиальной невидимости отрезка *ij*:

$$(x_i<x_л \;\&\; x_j<x_л) \vee (x_i>x_п \;\&\; x_j>x_п) \vee (y_i<y_н \;\&\; y_j<y_н) \vee (y_i>y_в \;\&\; y_j>y_в).$$

Полную тривиальную видимость отрезка *ab* определяет условие

$$(x_a>x_л \;\&\; x_b>x_л) \;\&\; (x_a<x_п \;\&\; x_b<x_п) \;\&\; (y_a>y_н \;\&\; y_b>y_н) \;\&\; (y_a<y_в \;\&\; y_b<y_в).$$

Неочевидной (нетривиальной) является полная невидимость отрезка *kl*, концы которого лежат по разные стороны как вертикальной, так и горизонтальной сторон окна. Отрезок *gh* расположен таким же образом, но является частично видимым. Очевидно, что только сравнением координат концов отрезка с положением сторон окна выявить полную невидимость невозможно.

Несмотря на простоту приведенных выше условий полной видимости или невидимости отрезков, они являются громоздкими и не наглядными. Коэн и Сазерленд (4) в 1968г. предложили алгоритм отсечения отрезка, в том числе способ определения полной тривиальной видимости или невидимости отрезков.

1001	1000	1010

0001	0000	0010
0101	0100	0110

Рис.6.2. Разбиение областей

На рис.6.2 показано, что экран по отношению к окну разбивается на девять областей, кодируемых четырьмя битами (выше, ниже, правее, левее окна). Области внутри окна соответствует код 0000b. Если оба кода концов отрезка равны нулю, отрезок целиком находится внутри окна. Если поразрядная конъюнкция кодов концов отрезка не равна нулю, отрезок является полностью невидимым. Если только один из кодов равен нулю, то отрезок частично видим (табл. 6.1).

Таблица 6.1.

Отрезок	Коды концов отрезка		*P1 & P2*	Результат
	P1	*P2*		
ab	**0000**	**0000**	0000	Полностью видим
ij (1)	0010	0110	0010	Полностью невидим
ij (2)	0101	0100	0100	Полностью невидим
ij (3)	0001	0101	0001	Полностью невидим
ij (4)	1001	1000	1000	Полностью невидим
cd	**0000**	0010	0000	Частично видим
ef	0001	**0000**	0000	Частично видим
gh	0001	1000	0000	Неизвестно
kl	1000	0010	0000	Неизвестно

Для построения частично видимого отрезка необходимо найти точку его пересечения границы окна. Для этого нужно решить четыре системы линейных уравнений, по одной для каждой границы окна.

Пусть отрезок *P1P2* имеет координаты концов (x_1, y_1) и (x_2, y_2) соответственно. Уравнение прямой имеет вид

$$y = ax+b.$$

Применительно к нашему случаю $a = (y_2 – y_1)/(x_2 – x_1)$, $b = y_1 – a\, x_1$ или $b = y_2 – a\, x_2$, b – точка пересечения прямой с осью ординат. Тогда

$$y = a(x – x_1)+ y_1 \text{ или } y = a(x – x_2)+ y_2.$$

Система уравнений для пересечения прямой с левой границей окна:

$$x = x_л ;$$

$$y = a(x – x_1) + y_1.$$

Решение: $y = a(x_л - x_1) + y_1$; ($a \neq \infty$, т.е. прямая не вертикальна). Левая граница окна имеет общую точку с отрезком, если $y > y_1$ и $y < y_2$. Аналогичным образом пересечение отрезка с правой границей окна:

$$y = a(x_п - x_1) + y_1.$$

Система уравнений для точки пересечения прямой с верхней границей окна:

$$y = y_в;$$

$$y = a(x - x_1) + y_1.$$

Решение: $y_в = a(x - x_1) + y_1$; $a(x - x_1) = y_в - y_1$; $x - x_1 = (y_в - y_1)/a$;

$x = x_1 + (y_в - y_1)/a$; ($a \neq \infty$, т.е. прямая не горизонтальна).

Верхняя граница окна имеет общую точку с отрезком, если $x > x_1$ и $x < x_2$.

Пересечение отрезка с нижней границей окна: $x = x_1 + (y_н - y_1)/a$.

6.3. Алгоритм деления отрезка средней точкой

Рассмотренные выше формулы для вычисления точек пересечения отрезка с границей окна достаточно простые, но в них присутствуют операции умножения и деления над действительными числами. Таким образом, аппаратная реализация таких простых действий, как отсечение отрезков прямых, уже потребует наличия в видеоконтроллере плавающей арифметики. Роберт Спрулл и Айвен Сазерленд (4) предложили делить отрезок пополам и рассматривать его половины как отдельные отрезки с помощью табл. 6.1. Если для нового отрезка не выполняется условие полной видимости или тривиальной невидимости, то он опять делится пополам до тех пор, пока не выродится в точку. Среднее число итераций n для отрезка длиной L, пересекающего границу окна (худший случай) в среднем равно

$$n = log_2 L.$$

Заметим, что для отрезка на растре его длина в пикселях равна разности координат по длинной стороне. Отрезок длиной 1024 пикселя выродится в точку всего через 10 итераций. В более благоприятных случаях, как на рис. 6.3, потребовалась всего одна итерация.

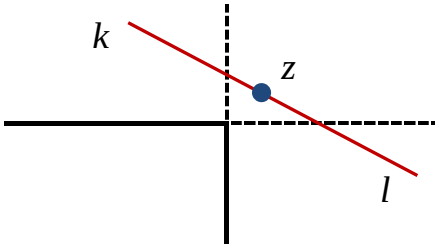


Рис.6.3.

Отрезок kl не идентифицируется как тривиально невидимый, но после деления средней точкой z отрезки kz и zl уже являются тривиально

невидимыми, следовательно весь отрезок *kl* также является тривиально невидимым. В других случаях, возможно, потребовалось бы большее число дихотомий (двоичных делений).

Рассмотрим теперь, как визуализируется частично видимый отрезок *gh* (рис. 6.4). Разделим его пополам точкой z_1. Обе половины отрезка gz_1 и z_1h также являются частично видимыми. Следующее деление точкой z_2 выделит отрезок gz_1 как полностью невидимый, а отрезок z_2z_1 опять потребует деления. Разделим пополам отрезок z_1h точкой z_3 получим полностью видимый отрезок z_1z_3. Еще через пару итераций мы получим видимую часть отрезка *gh* с пренебрежимо малой погрешностью. Таким образом, только с помощью операций деления координат отрезка пополам и сравнений мы можем найти части отрезков, которые должны визуализироваться. Напоминаем, что деление на два легко выполняется сдвигом вправо на один двоичный разряд.

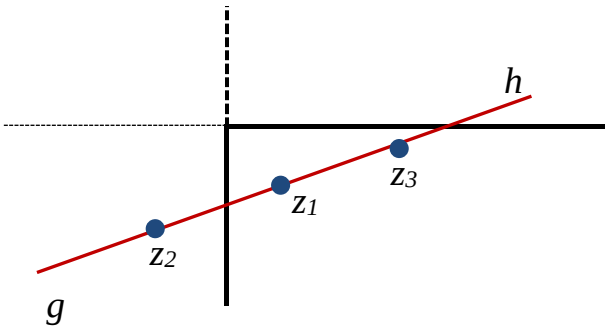


Рис.6.4.

ГЛАВА 7. СИНТЕЗ РЕАЛИСТИЧНЫХ ИЗОБРАЖЕНИЙ

7.1. Зачем нужна реалистичность

Компьютерная графика первоначально использовалась для построения диаграмм, чертежей и для других научных и технических задач, заменяя собой кульман конструктора для двумерных изображений и пенопластовые макеты для трехмерных. Первыми стали использовать реалистичные изображения разработчики компьютерных игр и тренажеров. В начале XXI века качество синтезированных изображений на обычных компьютерах (а не на специализированных графических станциях) достигло уровня, когда его стала признавать полиграфия, и это стало началом широкого применения компьютерной графики в рекламном бизнесе.

Еще одно применение компьютерной графики – мультипликация, производство которых всегда было очень дорогим занятием. Одним из первых фильмов с широким применением компьютерной визуализации был фильм «Кто подставил кролика Роджера», снятый в 1988 г. компаниями «*Touchstone Pictures*» и «*Amblin Entertainment*». В этом фильме комбинируются сцены с живыми актерами и компьютерная

мультипликация. Согласно законам жанра, мультипликация – изначально условное отображение реальных или вымышленных объектов. Несмотря на это, фильм «Кто подставил кролика Роджера» был самым дорогим фильмом своего времени, что может объясняться высокими затратами на компьютерную анимацию. Та же мультипликация в фильме Дж.Камерона «Аватар», снятого в 2009 г., выглядит намного реалистичнее.

Рис.7.1. Визуализация при проектировании дорог.

Применение фотореалистичных изображений не ограничивается индустрией компьютерных игр или рекламным бизнесом. Новые возможности, предоставляемые компьютерной графикой, позволяют поднять на новый уровень проектирование. Если 40 лет назад большим достижением была возможность проектирования дорог с визуализацией результата с точки зрения водителя, проезжающего по еще несуществующей дороге (рис.7.1), то сейчас можно для дорог проектировать их освещение. Для проектируемого автомобиля можно проверить, как будет отражаться свет от стекол, приборов и других предметов в салоне, каким будет обзор, в т.ч. через зеркала.

Рис. 7.2. Промежуточная и финальная визуализация.

На рис. 7.2. показаны два варианта визуализации. Первый (промежуточный) – получен из программы, с помощью которой проектировалось здание. Второй вариант – финальная визуализация, во время которой добавляются мелкие детали, элементы экстерьера и освещение. С одной стороны, это позволяет лучше преподнести проект заказчику, с другой – оценить, как новый объект будет вписываться в окружающую застройку, что весьма актуально для строительства в исторической части города.

7.2. Особенности человеческого зрения

Все, что мы видим, это игра света, прямого или отраженного, собираемого на сетчатке глаза. Понятие цвета любого объекта не существует в отрыве от его освещения и от свойств человеческого зрения. Таким образом, синтез реалистичных изображений должен выполняться путем моделирования световых лучей с учетом психофизиологических особенностей человеческого зрения.

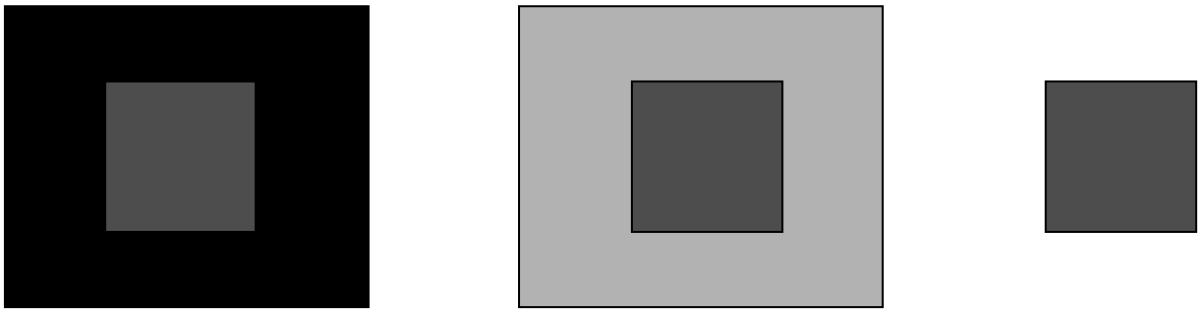


Рис. 7.3. Иллюстрация обмана зрения.

Пример на рис. 7.3 показывает, что восприятие яркости является относительным. Три маленьких квадрата на рисунке кажутся разным по яркости. Между тем их яркость одинакова. Если существуют люди с абсолютным слухом, способные распознавать отдельные ноты в отрыве от заданной тональности, то можно смело утверждать, что не существует людей, которые могли бы распознавать уровни яркости и цвета без опорных точек. Глаз человека комбинирует в себе два устройства – черно-белый прибор почти ночного видения и цветной фотоаппарат. Диапазон чувствительности глаза к яркости равен приблизительно 10^{10}, но в пределах одной картинки глаз различает только 100-150 уровней. Чтобы перейти в другой диапазон яркостей глаз должен привыкнуть к новому освещению.

Все это возможно благодаря тому, что на сетчатке глаза расположены два типа световых рецепторов, палочки и колбочки. Палочки обеспечивают чувствительность к низким уровням яркости без различения цветов, а колбочки – распознавание цветов и мелких деталей. Цветовое зрение также может адаптироваться к оттенкам освещения. В результате глаз, выражаясь языком фотографов, автоматически выставляет баланс белого.

Помимо особенностей восприятия цветов, существуют также особенности восприятия формы. Трехмерные объекты отображаются на двумерной сетчатке, а мозг человека восстанавливает трехмерную модель.

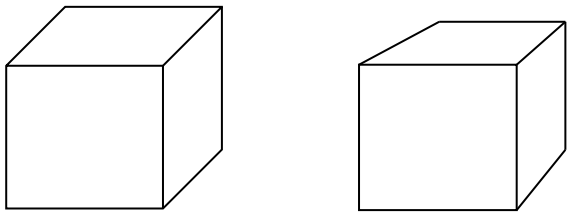


Рис. 7.4. Аксонометрическая и перспективная проекции

На рис.7.4 показаны известные из школьного курса черчения

аксонометрическая и перспективная проекции куба. Первая является умозрительной, поскольку так можно увидеть куб только с бесконечного расстояния. В реальности задняя грань куба находится дальше от наблюдателя и, следовательно, имеет меньшие угловые размеры. Насколько меньшие, зависит от размера объекта и расстояния до него. Мозг человека, а точнее, зрительный центр, обрабатывает эту информацию и восстанавливает модель куба. Однако, зрение человека легко обмануть. Если посмотреть на удаленный объект через бинокль, то он будет казаться гораздо ближе, чем на самом деле, и задние грани кубических объектов будут казаться больше передних (рис. 7.5).

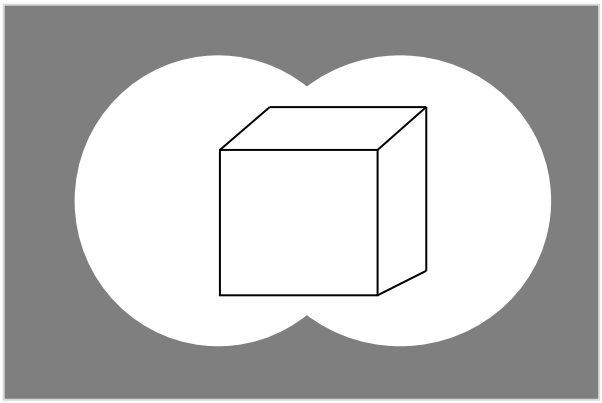


Рис. 7.5. Иллюзия искажения куба в бинокле

Если, однако, смотреть в бинокль достаточно долго, то глаз привыкает к такой перспективе и показанная выше иллюзия исчезает. Таким образом, зрительный образ возникает в мозгу человека путем достаточно сложной обработки знаний. Чтобы глаз быстро настроился на перспективу, человек должен распознать наблюдаемые предметы, извлечь из памяти их истинную форму и настроить взгляд. Есть еще более экзотические примеры. Как утверждает главный офтальмолог России В. Нероев, пигмеи, живущие в густых лесах, попав на открытое пространство, принимают пасущегося вдали быка за насекомое («АиФ», №5, 2011г.). Подобная проблема возникает при просмотре 3D изображений на современных телевизорах. Если в 3D кинотеатрах экран расположен достаточно далеко от зрителя, то телевизор смотрят на близком расстоянии, и появляется проблема аккомодации глаза. Трехмерная картинка, создается в мозгу с помощью специальных затворных очков, когда левый глаз видит одну картинку, а правый – другую. Картинки чередуются с высокой скоростью, а очки перекрывают доступ то к левому, то к правому глазу. Благодаря этому, параллакс ближних предметов отличается от параллакса дальних, что создает стереоэффект. Между тем глаз должен всегда фокусироваться на расстояние экрана, и возникает диссонанс: объекты находятся визуально на одном расстоянии, а глаз фокусируется на другое расстояние.

Этот эффект может вызывать головную боль и быструю утомляемость вследствие того, что мозг перегружен работой по исправлению постоянно возникающих противоречий.

Кстати, вид поля зрения через бинокль вовсе не такой, как показано на рис.7.5. Это всего лишь стереотип или условность, чтобы показать, что изображение получено с помощью бинокля, а между тем поле зрения имеет форму круга независимо от того, одним глазом смотрит наблюдатель или двумя.

Исходя из сказанного, можно сделать вывод, что задача синтеза реалистичного изображения заключается в том, сформировать на экране такое изображение, которое будет отображено на сетчатке человека точно так же, как бы это было получено от реального объекта и чтобы это изображение не было противоречивым.

7.3. Модель освещения

Как показано выше, мы видим все объекты исключительно благодаря свету. Если объект сам является источником света, мы видим его собственный свет. В противном случае объект виден за счет его взаимодействия с источниками света. Объект может пропускать или отражать свет. Если объект пропускает свет, то происходит его фильтрация, поскольку чаще всего часть спектра поглощается, а также происходит преломление. Отражение света может быть зеркальным или диффузным.

Зеркальное отражение происходит по хорошо известному закону «угол падения равен углу отражения», или, более строго, падающий и отражённый лучи лежат в одной плоскости с нормалью к отражающей поверхности в точке падения, и эта нормаль делит угол между лучами на две равные части (рис. 7.6).

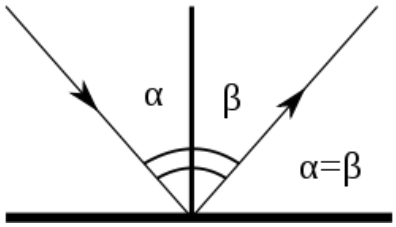


Рис.7.6. Идеальное зеркальное отражение

Если поверхность является шероховатой, то происходит не зеркальное, а диффузное отражение, т.е. свет отражается во все стороны равномерно (рис. 7.7).

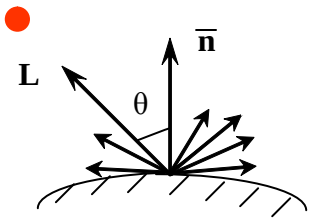


Рис. 7.7. Диффузное отражение.

Здесь L – источник света, $\overline{n}$ – нормаль к поверхности, Θ – угол между направлением на источник света и нормалью к поверхности.

Шероховатость определяется относительно длины волны света, и одна и та же поверхность может быть матовой, диффузно отражающей для ультрафиолета, но зеркально гладкой для инфракрасного света.

Интенсивность отраженного света определяется следующим образом:

$$I = I_l \cdot k_d \cdot \cos\theta$$

где I – интенсивность отраженного света;
I_l – интенсивность падающего света от источника;
k_d – коэффициент диффузного отражения, $0 \le k \le 1$;
θ - угол между направлением света и нормалью к поверхности, $0 \le \theta \le \pi/2$.

При создании модели освещения даже с одним источником мы сталкиваемся с тем, что возникает множество вторичных источников освещения за счет отражения света от окружающих предметов. К тому же свет отраженный от больших поверхностей, не является точечным, что приводит к необходимости проведения огромного числа вычислений. Для упрощения расчетов в таких случаях используют дополнительный источник рассеянного света (фоновую освещенность), одинакового для всех точек, и интенсивность отраженного света определяется следующей формулой.

$$I = I_a k_a + I_l \cdot k_d \cdot \cos\theta\ ,$$

где I_a – интенсивность рассеянного света, k_a – коэффициент диффузного отражения рассеянного света, $0 \le k_a \le 1$.

Для вычисления I нужно знать, какое количество света доходит от источника до объекта. Как известно, свет распространяется от источника во все стороны, следовательно, его интенсивность обратно пропорционально квадрату расстояния. Однако, практика показывает, что при моделировании близко расположенных источников квадратичная

зависимость дает слишком сильный эффект, и линейное затухание света показывает достаточно адекватную картину. Поэтому формула для диффузного отражения может быть представлена следующим образом.

$$I = I_a k_a + \frac{I_l k_d \cos\theta}{d + k},$$

где d - расстояние от источника до объекта;
k – произвольная постоянная (калибровочный параметр).

Зеркальное отражение гораздо сложней диффузного хотя бы потому, идеальное зеркальное отражение встречается редко (рис.7.8).

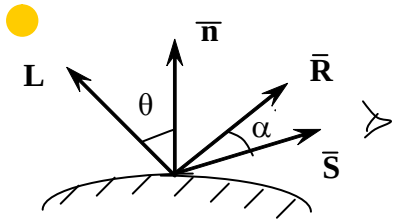


Рис.7.8. Реальное зеркальное отражение.

Для моделирования зеркального отражения используется эмпирическая модель Фонга:

$$I_s = I_l W(\theta, \lambda) \cos^n \alpha$$

где $W(\theta, \lambda)$ – кривая отражения, представляющая отношение зеркально отраженного света к падающему, как функцию угла падения θ и длины волны λ, n – степень, аппроксимирующая пространственное распределение зеркально отраженного света.

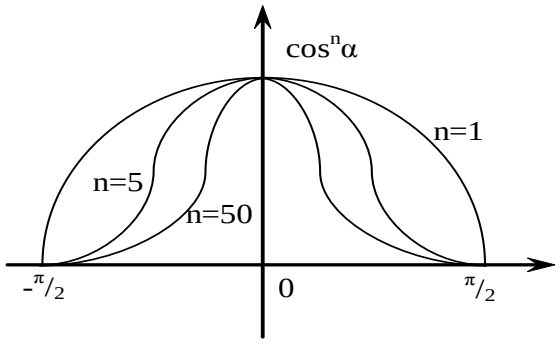


Рис.7.9. Кривая отражения.

Чем больше величина *n*, тем ближе отражение к идеальному. Функция

$W(\theta,\lambda)$ является довольно сложной (рис.7.10),

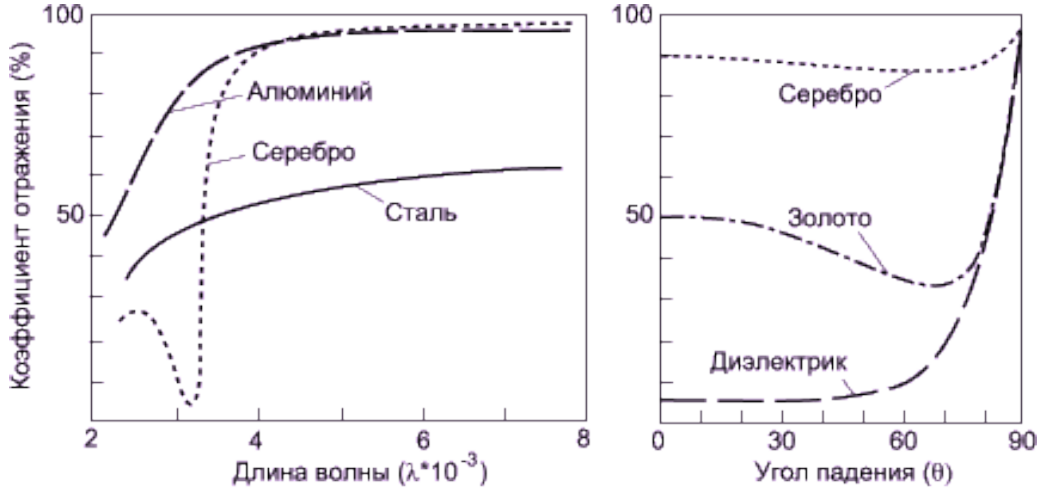


Рис.7.10. Кривая отражения для различных материалов.

поэтому ее обычно заменяют константой, k_s, значение которой выбирается эмпирически. Таким образом, функция закраски точки с учетом диффузного и зеркального отражения от одного источника света может выглядеть следующим образом:

$$I = I_a k_a + \frac{I_l}{d+k}(k_d \cos\theta + k_S \cos^n \alpha)\,.$$

В векторной форме

$$\cos\theta = \frac{\overline{n}\cdot\overline{L}}{|n||L|} = \hat{n}\hat{L}$$

$$\cos\alpha = \frac{\overline{R}\cdot\overline{S}}{|R||S|} = \hat{R}\hat{S}$$

Здесь $\overline{n}$ – нормаль к поверхности;
$\overline{L}$ – направление на источник;
$\overline{R}$ – вектор, определяющий направление отраженного луча;
S - вектор, определяющий направление наблюдения (все вектора единичные).
В векторной форме формула закраски пикселя от одного источника выглядит следующим образом:

$$I = I_a k_a + \frac{I_l}{d+k}(k_d(\overline{n}\cdot\overline{L}) + k_S(\overline{R}\cdot\overline{S})^n)$$

Если источников несколько, то вместо второго слагаемого должна

быть сумма по всем источникам:

$$I = I_a k_a + \sum_l \frac{I_l}{d_l + k_l}(k_d(\bar{n} \cdot \bar{L_l}) + k_S(\bar{R_l} \cdot \bar{S})^n)$$

Наконец, поскольку отражение разных цветов происходит по-разному, и именно поэтому мы видим предметы цветными, то коэффициенты k_a, k_d и k_s устанавливаются для каждого цвета отдельно, и освещенность точки вычисляется раздельно для каждого из основных цветов.

Рассмотрим процесс вычисления освещенности на простом примере (рис.7.11).

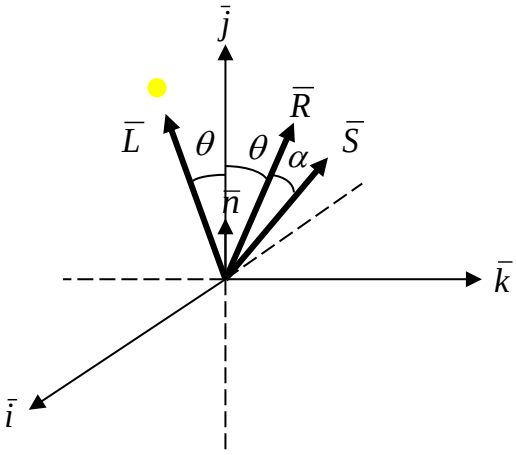


Рис. 7.11. Пример вычисления освещенности точки.

Пусть точка располагается в центре системы координат *i, j, k*, причем вектор нормали $\bar{n} = \bar{j}$, т.е. поверхность в центре касательна горизонтальной плоскости *jk*. Пусть вектор падающего света

$$\bar{L} = -\bar{i} + 2\bar{j} - \bar{k},$$

а вектор наблюдателя

$$\bar{S} = -\bar{i} + 1{,}5\bar{j} + 0{,}5\bar{k}.$$

Сначала найдем вектор отражения

$$\bar{R} = \bar{i} + 2\bar{j} + \bar{k}.$$

Пусть $I_a = 1$, $I_l = 10$, т.е. интенсивность света от источника в 10 раз выше интенсивности рассеянного света, а поверхность блестящая, и свет отражается, в основном, зеркально, а также *d* = 0, *k* = 1.

$$k_s = 0{,}8,\ k_a = k_d = 0{,}15,\ n = 5,\ k_s + k_d = 0.95.$$

Находим углы Θ и α.

$$\hat{n}\cdot\hat{L}=\frac{\bar{n}\cdot\bar{L}}{|n||L|}=\frac{\bar{j}\cdot(-\bar{j}+2\bar{j}-\bar{k})}{1\cdot\sqrt{-1^2+2^2+(-1)^2}}=\frac{2}{\sqrt{6}};$$

$$\theta=\cos^{-1}(\frac{2}{\sqrt{6}})=35,26°;$$

$$\hat{R}\cdot\hat{S}=\frac{\bar{R}\cdot\bar{S}}{|R||S|}=\frac{(\bar{i}+2\bar{j}+\bar{k})(\bar{i}+1.5\bar{j}+0.5\bar{k})}{\sqrt{1^2+2^2+1^2}\cdot\sqrt{1^2+1.5^2+0.5^2}}=\frac{4.5}{\sqrt{6}\cdot\sqrt{3.5}}=\frac{4.5}{\sqrt{21}};$$

$$\alpha=10.89°.$$

Теперь можно вычислить интенсивность цвета точки:

$$I=I_a k_a+\sum_l \frac{I_l}{d_l+k_l}(k_d(\bar{n}\cdot\overline{L_l})+k_S(\overline{R_l}\cdot\bar{S})^n)$$

$$I=1\cdot 0,15+\frac{10}{0+1}\left[0,15\cdot\left(\frac{2}{\sqrt{6}}\right)+0,8\cdot\left(\frac{4,5}{\sqrt{21}}\right)^5\right]=8,65.$$

При изменении положении наблюдателя

$$\bar{S}=\bar{i}+1,5\bar{j}-0,5\bar{k};$$

$$\hat{R}\cdot\hat{S}=\frac{3,5}{\sqrt{21}};\qquad \alpha=40,2°;$$

$$I=3,45.$$

7.4. Нахождение нормали к поверхности

Как показано выше, для вычисления интенсивности света, отражаемого от поверхности, необходимо располагать нормалями к поверхности в каждой отображаемой точке. Вычисление нормалей – достаточно сложная операция.

Возможны следующие варианты задания поверхности:

1. При аналитическом задании поверхности нормаль вычисляется непосредственно, поскольку известны уравнения поверхностей. В самом простом случае для плоскости, уравнение которой

$$a_1 i+ b_1 j+ c_1 k +d= 0,$$

вектор нормали $\bar{n} = a_1 i+ b_1 j+ c_1 k.$

2. Во многих задачах поверхность представляется полигональной аппроксимацией, тогда зная уравнение плоскости каждого полигона, можно найти уравнение внешней нормали.
3. Во многих алгоритмах удаления невидимых линий и поверхностей используются только ребра или вершины; тогда находится приближенное значение нормали на ребрах или вершинах.

Отображение ребер объемных фигур является важной составляющей синтеза реалистичных изображений. В реальности ребра имеют ненулевую ширину, но моделировать их форму достаточно сложно, поэтому используются приближенные методы. Например, ребро, образованное пересечением плоскостей, задаваемых уравнениями $a_1i+ b_1j+ c_1k +d_1= 0$ и $a_2i+ b_2j+ c_2k +d_2= 0$, может быть представлено как плоскость шириной в один пиксель, нормаль к которой равна векторной сумме нормалей к каждой из граней:

$$\bar{n}_{12} = (a_1 + a_2)\cdot\bar{i} + (b_1 + b_2)\cdot\bar{j} + (c_1 + c_2)\cdot\bar{k}.$$

По сути это означает, что грани геометрически идеальной фигуры как бы слегка обработаны напильником для сглаживания острых кромок. Аналогично можно определить освещенность вершины, в которой сходятся три грани:

$$\bar{n}_{123} = (a_1 + a_2 + a_3)\cdot\bar{i} + (b_1 + b_2 + b_3)\cdot\bar{j} + (c_1 + c_2 + c_3)\cdot\bar{k}.$$

Если заданы ребра, например $\bar{V}_1\bar{V}_2, \bar{V}_1\bar{V}_4, \bar{V}_1\bar{V}_5$, то нормаль к вершине можно определить как векторную сумму ребер:

$$\bar{n}_{V_1} = \bar{V}_1\bar{V}_2 \otimes \bar{V}_1\bar{V}_4 + \bar{V}_1\bar{V}_5 \otimes \bar{V}_1\bar{V}_2 + \bar{V}_1\bar{V}_4 \otimes \bar{V}_1\bar{V}_5.$$

Пример. Имеется усеченная пирамида (рис. 7.12)

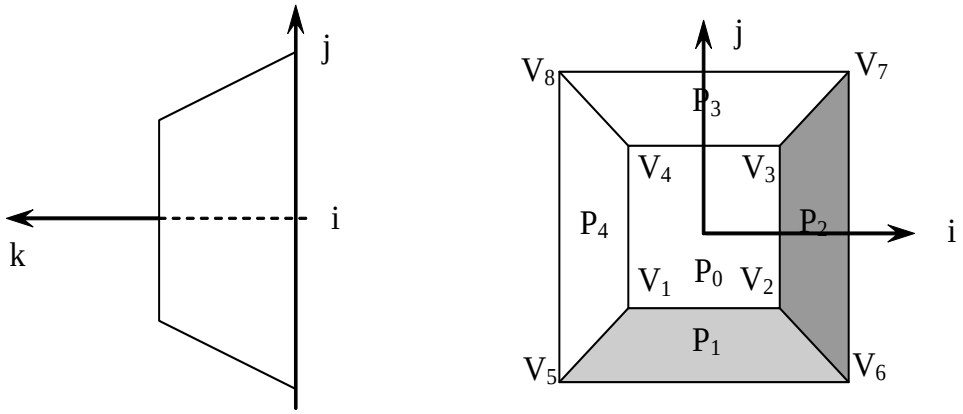


Рис. 7.12. Пример вычисления нормали.

с координатами вершин

V_1 (-1; -1; 1);	V_2 (1; -1; 1);
V_3 (1; 1; 1);	V_4 (-1; 1; 1);
V_5 (-2; -2; 0);	V_6 (2; -2; 0);
V_7 (2; 2; 0);	V_8 (-2; 2; 0).

Запишем уравнения плоскостей для граней P_0, P_1, P_4:

$P_0 : k - 1 = 0;$

$P_1 : -j + k - 2 = 0;$

$P_4 : -i + k - 2 = 0.$

Найдем приближенную нормаль в точке V_1, усредняя нормали к окружающим многоугольникам

$$\bar{n}_{V_1} = (a_0 + a_1 + a_4)\cdot\bar{i} + (b_0 + b_1 + b_4)\cdot\bar{j} + (c_0 + c_1 + c_4)\cdot\bar{k} = -\bar{i} - \bar{j} + 3\bar{k}.$$

Абсолютная величина нормали может быть найдена следующим образом:

$$|n_{V_1}| = \sqrt{(-1)^2 + (-1)^2 + (+3)^2} = \sqrt{11},$$

а единичная нормаль –

$$\frac{\bar{n}_{V_1}}{|n_{V_1}|} = -\frac{1}{\sqrt{11}}\bar{i} - \frac{1}{\sqrt{11}}\bar{j} + \frac{3}{\sqrt{11}}\bar{k} = -0,3\bar{i} - 0,3\bar{j} + 0,9\bar{k}.$$

Теперь найдем выражение нормали к той же точке через векторное произведение ребер, сходящихся в вершине V_1.

$$\bar{V_1V_2} \otimes \bar{V_1V_4} = \begin{vmatrix} i & j & k \\ 1+1 & -1+1 & 1-1 \\ -1-1 & 1+1 & 1-1 \end{vmatrix} = \begin{vmatrix} i & j & k \\ 2 & 0 & 0 \\ 0 & 2 & 0 \end{vmatrix} =$$

$$= (0\cdot 0 - 0\cdot 2)\bar{i} + (0\cdot 0 - 2\cdot 0)\bar{j} + (2\cdot 2 - 0\cdot 0)\bar{k} = 4\bar{k}.$$

Аналогичным образом вычисляем

$$\bar{V_1V_5} \otimes \bar{V_1V_2} = -2\bar{j} + 2\bar{k}$$

$$\bar{V_1V_4} \otimes \bar{V_1V_5} = -2\bar{i} + 2\bar{k}$$

Усредняя векторное произведение, получим нормаль в точке V_1:

$$\bar{n}_{V_1} = -2\bar{i} - 2\bar{j} + 8\bar{k}.$$

Абсолютная величина нормали

$$|\bar{n}_{V_1}| = \sqrt{(-2)^2 + (-2)^2 + (+8)^2} = \sqrt{72},$$

и единичная нормаль будет выглядеть следующим образом:

$$\frac{\bar{n}_{V_1}}{|n_{V_1}|} = -0.24\bar{i} - 0.24\bar{j} + 0.94\bar{k}$$

Заметим, что значения нормали, вычисленные двумя способами, не совпадают.

7.5. Отображение прозрачности

Поскольку прозрачные объекты имеют плотность, отличающуюся от плотности воздуха, происходит преломление лучей по закону Снеллиуса (рис. 7.13): падающие и преломленные лучи лежат в одной плоскости, а углы падения и преломления связаны формулой:

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

где η_1 – показатель преломления среды, из которой свет падает на границу раздела; Θ_1 – угол падения света – угол между падающим на поверхность лучом и нормалью к поверхности; η_2 – показатель преломления среды, в которую свет попадает, пройдя границу раздела; Θ_2 – угол преломления света — угол между прошедшим через поверхность лучом и нормалью к поверхности.

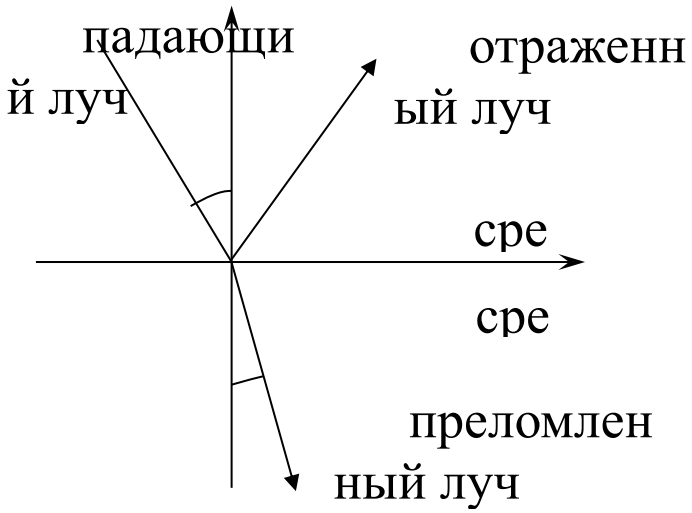


Рис. 7.13. Преломление света.

Таким образом, интенсивности преломленного света в прозрачных объектах вычисляются аналогично отражению, только вместо закона отражения применяется закон преломления.

Однако расчет освещенности здесь осложняется следующими факторами.

Во-первых, показатели преломления зависят от длины волны света, следовательно, каждый луч, не относящийся к основным цветам красному, зеленому или синему, будет разлагаться на составляющие. Например, желтый луч в реальности будет отклоняться на один угол, но его реализация в виде смеси красного и зеленого даст два угла преломления.

Во-вторых, как снаружи, так и внутри прозрачного объекта может происходить отражение луча: часть луча отражается, остальная часть отражается. Многократное внутреннее отражение может существенно усложнить задачу формирования изображения прозрачного объекта.

В-третьих, внутри прозрачного объекта световой луч частично поглощается, и степень поглощения зависит от пройденного расстояния.

На практике приходится ограничивать число внутренних отражений, чтобы выполнить синтез изображения в разумные сроки.

7.6. Отображение фактуры поверхности

При синтезе реалистичных изображений возникает необходимость отображать не только гладкие поверхности, но и поверхности, имеющие либо рельеф, либо неоднотонный цвет (узор), либо и то и другое вместе. Всё это называется фактурой поверхности. Поверхности с фактурой распространены гораздо больше, чем гладкие поверхности. Прямое воспроизведение фактуры, например, ткани, путем отображения каждой нити со всеми переплетениями не представляется возможным.

Решением может служить имитация фактуры одним из двух способов. Первый способ предполагает создание на поверхности неровностей, шероховатостей, которые реализуются путем внесения возмущений в параметры, задающие поверхность. Таким путем можно имитировать как регулярные, так и случайные фактуры.

Второй способ заключается в нанесении на поверхность рисунка - текстуры. В качестве такого рисунка могут использоваться как синтезированные изображения, так и импортируемые. В простейшем случае рисунок накладывается на плоскую поверхность, скажем, при отображении картины, висящей на стене. В более сложных случаях требуется «натягивать» плоскую картинку на объемный объект, например, карту мира на шар, чтобы получить глобус.

Задача наложения фактуры на поверхность заключается в преобразовании координат каждой точки двумерной текстуры в

координаты (двумерные или трехмерные) поверхности объекта.

Пусть текстура существует в ортогональной двумерно системе координат (*U*,*W*), а объект – в ортогональной трехмерной системе координат (*X*,*Y*,*Z*). Тогда наложение текстуры на объект заключается в установлении однозначного соответствия каждой точки объекта (*x*,*y*,*z*) точке текстуры (*u*,*w*):

$$x = f_x(u,w)\ ;\ y = f_y(u,w);\ z = f_z(u,w).$$

Рассмотрим широко используемый в учебниках пример отображения текстуры на шар. При всей кажущейся сложности это несложное преобразование координат, поскольку шар – это простое геометрическое тело.

Пусть текстура – квадрат со сторонами (1,1) в системе координат (*u*,*w*) (рис. 7.14), а шар имеет радиус *r* и существует в системе координат (*x*,*y*,*z*). Уравнение сферы с центром в центре координат имеет следующий вид:

$$x^2 + y^2 + x^2 = r^2.$$

Из этого уравнения можно вывести формулы для вычисления каждой координаты по двум другим, однако, здесь гораздо проще ввести в качестве промежуточной полярную систему координат (*Θ*,*Φ*,*R*), в центре которой находится центр шара.

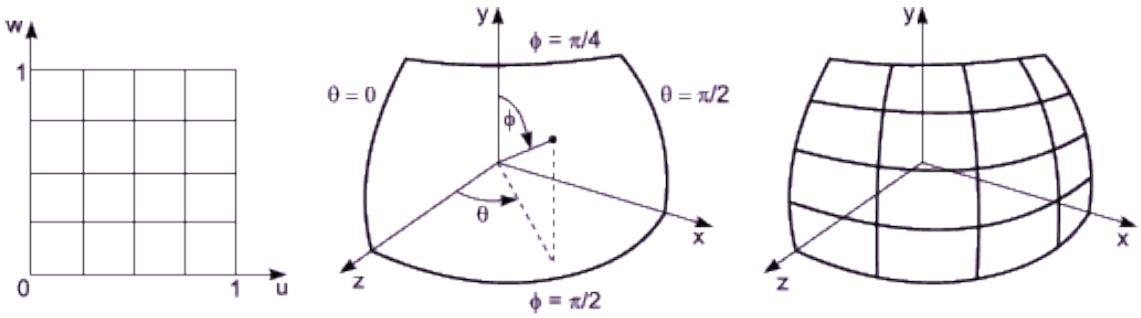


Рис.7.14. Наложение сетки на сферу.

Полярные координаты каждой точки сферы определяются двумя углами и радиусом (*θ*,*φ*,*r*) и преобразуются в ортогональные координаты следующими формулами:

$$x = r\sin\theta\sin\phi,$$
$$y = r\cos\phi,$$
$$z = r\cos\theta\sin\phi.$$

Пусть текстура размером (1,1) накладывается на октант сферы

$$0 \le \theta \le \frac{\pi}{2}, \quad \frac{\pi}{4} \le \phi \le \frac{\pi}{2}.$$

В этом случае левая нижняя точка текстуры отображается на левую нижнюю (на рисунке) точку сферы:

$$u = 0, \ \mathrm{w} = 0 \ \rightarrow \ \theta = 0, \ \phi = 0.$$

Соответственно, правая нижняя точка текстуры отображается на правую нижнюю (на рисунке) точку сферы:

$$u = 1, \ \mathrm{w} = 0 \ \rightarrow \ \theta = \frac{\pi}{2}, \ \phi = 0.$$

Верхний левый и верхний правый углы текстуры отображаются на одну и ту же точку сферы – «северный полюс»:

$$u = 0, \ \mathrm{w} = 1 \ \rightarrow \ \theta = 0, \ \phi = \frac{\pi}{2};$$

$$u = 1, \ \mathrm{w} = 1 \ \rightarrow \ \theta = \frac{\pi}{2}, \ \phi = \frac{\pi}{2}.$$

Заметим, что угол θ соответствует долготе на глобусе и не имеет смысла на полюсе. Тем не менее, мы это угол установили, поскольку он должен использоваться для вычисления функции преобразования.

Самое простое преобразование координат – линейное:

$$\theta = Au + B;$$
$$\phi = Cw + D,$$

где *A,B,C,D* – коэффициенты, значения которых могут быть получены из назначенных нами координат граничных точек. Кстати, привязывать текстуру к поверхности не обязательно по граничным точкам. Например, при наложении портрета на модель головы (матрешка) такими точками могут быть зрачки, уголки губ и т.д.

При изменении u от 0 до 1 φ меняется от 0 до π/2, и при изменении w от 0 до 1 θ меняется от 0 до π/2. Следовательно,

$$D = 0; C = \frac{\varphi_2 - \varphi_1}{u_2 - u_1} = \frac{\frac{\pi}{2} - 0}{1 - 0} = \frac{\pi}{2}.$$

Аналогично, B =0, A= π/2. Таким образом, получаем

$$\theta = \frac{\pi}{2} u; \phi = \frac{\pi}{2} w.$$

Теперь можно переходить от полярных координат к ортогональным и

вычислять (x,y,z).

Таким образом, синтез реалистичных изображений представляет собой сложную вычислительную задачу, причем сложность этой задачи постоянно возрастает с увеличением разрешающей способности устройств отображения информации. Для решения этой задачи используются приближенные методы и имитации, позволяющие создать у наблюдателя иллюзию реального объекта.

ГЛАВА 8. МОДЕЛЬ ОСВЕЩЕНИЯ С ТРАССИРОВКОЙ ЛУЧЕЙ

8.1. Принцип трассировки лучей

Как было показано в предыдущей главе, мы видим окружающий мир за счет света, испускаемого источниками, преломляемого и отражаемого предметами. Кроме этого возможны другие преобразования света от источников, например фотолюминесценция – преобразование света одной длины волны в другую длину. Существует также явление, непосредственно влияющее на компьютерный синтез изображений – интерференция, когда регулярная структура поверхности отображаемого объекта накладывается на растр устройства отображения. Указанные явления представляют собой отдельную проблему и здесь рассматриваться не будут.

Для отображения реалистичной картинки необходимо трассировать лучи, исходящие из всех источников, и попадающие в глаз наблюдателя сквозь экран. Именно яркости каждого луча и должна соответствовать яркость каждого пикселя на экране. Здесь сразу возникает вопрос, о каком количестве лучей идет речь. Очевидно, что в природе никаких лучей не существует, а свет распространяется согласно корпускулярно-волновой теории Луи де Бройля. Количество условных лучей испускаемых источниками очень велико, поскольку далеко не все они попадают в глаз наблюдателя (рис.8.1).

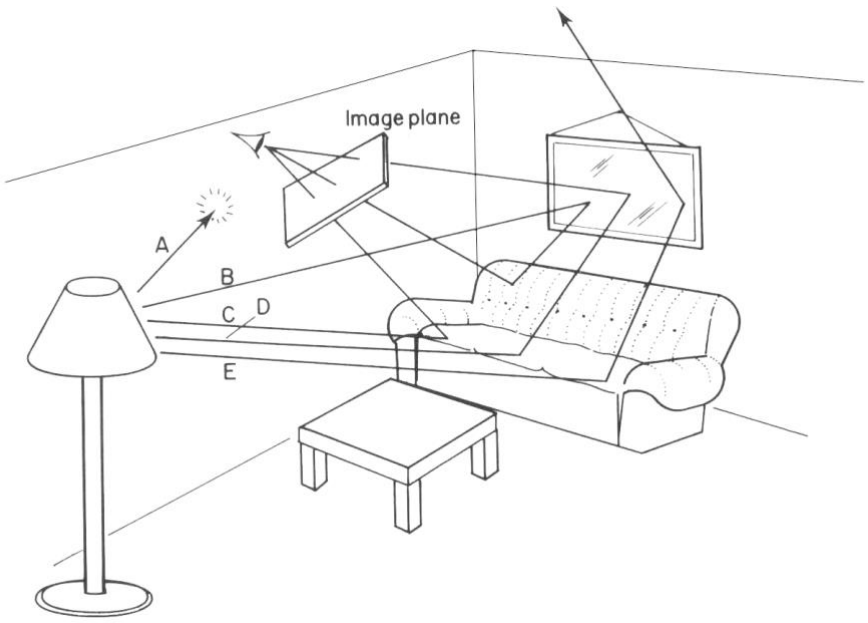


Рис.8.1. Прямая трассировка лучей.

В связи с этим задачу трассировки можно существенно упростить, заменив прямую трассировку на обратную. Иными словами, должны трассироваться зрительные лучи, исходящие из глаза наблюдателя,

проходящие сквозь экран и заканчивающиеся источником света (рис. 8.2).

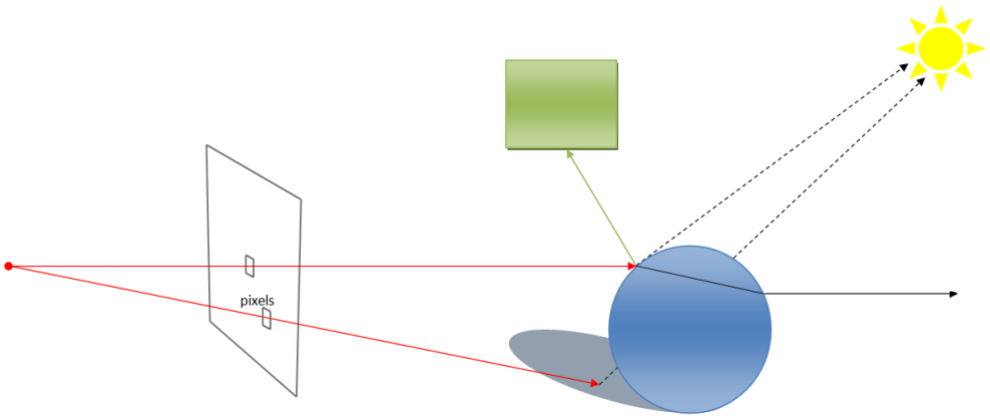


Рис. 8.2. Обратная трассировка лучей.

В таком случае количество лучей, которые должны трассироваться, равно числу пикселей на экране. Это число большое, но все же конечное. Правда, в ходе трассировки луч может разделяться. Так, на рис. 8.2 горизонтальный луч частично зеркально отражается от сферы, частично отражается диффузно, а кроме того преломляется. Прозрачные объекты имеют несколько точек пересечения зрительного луча. Трассировка луча должна заканчиваться, когда луч или его потомки достигают каждого источника света. Поскольку в результате многократных отражений луч затухает, обычно ограничиваются некоторым числом отражений или преломлений.

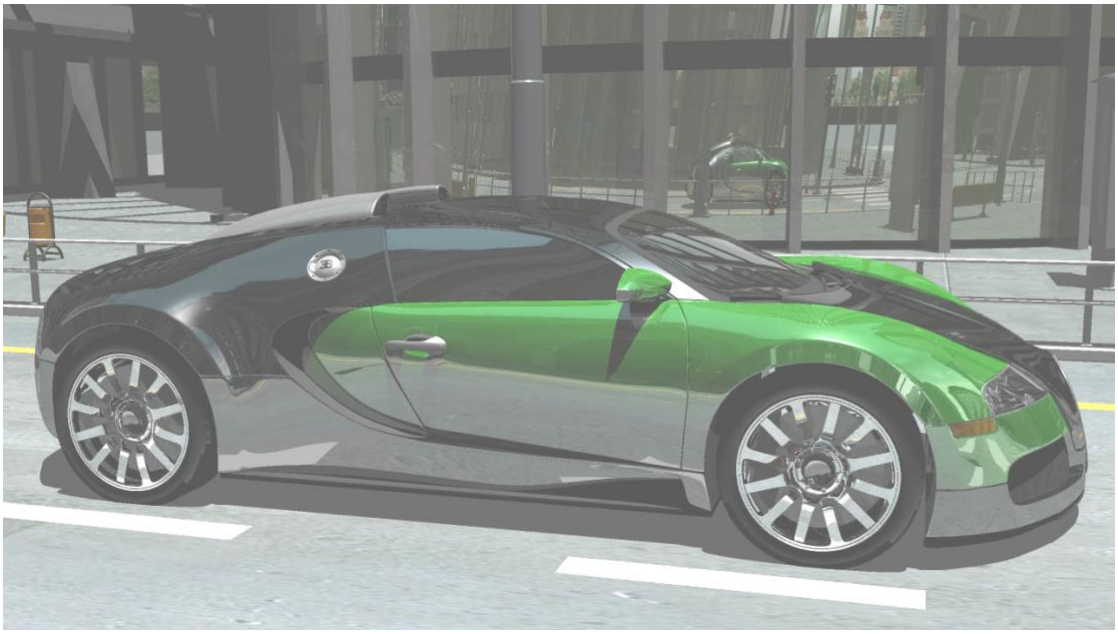


Рис.8.3. Пример изображения, синтезированного методом трассировки лучей.

На рис. 8.3. приведен пример изображения, созданного методом трассировки лучей. Здесь видны отражения одних объектов на другие, а также отражения объекта на самого себя, в частности, боковое зеркало автомобиля отражается в его правой двери. Если рассматривать эту картинку в хорошем качестве (http://www.thg.ru/graphic/ray_tracing_rasterization/ray_tracing_rasterization_screenshots_6_1.html), то его искусственность выдает только отсутствие оператора в отражении на витрине здания на заднем плане.

Метод трассировки лучей имеет следующие преимущества по сравнению с растровыми методами синтеза изображений:

1. Лучшее по сравнению с растеризацией отображение прозрачных и зеркальных объектов.
2. Расчет теней не требует сложных алгоритмов.
3. Гораздо проще выполняется отображение криволинейных объектов. Если для растеризации необходима математическая модель криволинейной поверхности, то при трассировке достаточно найти пересечение луча с объектом, который может быть задан разными способами. Это напоминает графо-аналитические методы решения в геометрии.

Однако, метод трассировки лучей не лишен недостатков, среди которых можно выделить следующие:

1. Основной недостаток – высокая сложность алгоритмов. Главное отличие метода трассировки лучей от растеризации заключатся в обработке отраженных лучей, а отражение лучей происходит в разные стороны, т.е. не обладают когерентностью, и в этом случае невозможно применять кеширование и другие методы ускорения расчетов. При растеризации все пиксели полигона обрабатываются одинаково или почти одинаково.
2. Метод не учитывает размеры лучей. В глазу наблюдателя луч имеет нулевое сечение, а проходя через экран, приобретает размер пикселя. При значительных удалениях луч существенно расширяется. Однако, алгоритмы трассировки лучей имеют дело с идеальным лучом, проходящим через центр пикселя. При отображении наклонных линий возможен эффект лесенок. Единственная технология, позволяющая устранить это эффект – суперсэмплинг, трассировка большего числа лучей, чем имеется пикселей.
3. Архитектуры графических процессоров (GPU) ориентированы на рендеринг полигонов, что обуславливает большую производительность растеризации.

8.2. Алгоритм трассировки лучей

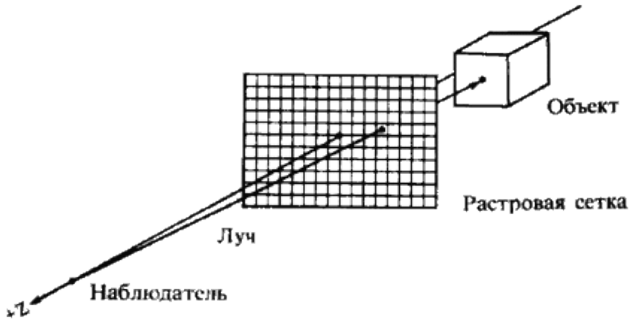


Рис. 8.4. К алгоритму трассировки лучей.

Каждый луч, исходящий от наблюдателя, походит через центр пикселя на растре до сцены. Траектория каждого луча отслеживается, чтобы определить, какие именно объекты сцены, если таковые существуют, пересекаются с данным лучом. Необходимо проверить пересечение каждого объекта сцены с каждым лучом. Если луч пересекает объект, то определяются все возможные точки пересечения луча и объекта. Можно получить большое количество пересечений, если рассматривать много объектов. Эти пересечения упорядочиваются по глубине. Пересечение с максимальным значением координаты z представляет видимую поверхность для данного пикселя. Атрибуты этого объекта используются для определения характеристик пикселя.

Таким образом, основным действием в алгоритме трассировки лучей является нахождение точки пересечения луча с объектом. Размерность этой задачи можно существенно сократить, ограничив пространство поиска простой фигурой, параллелепипедом или сферой (рис. 8.5), в которую вписан объект.

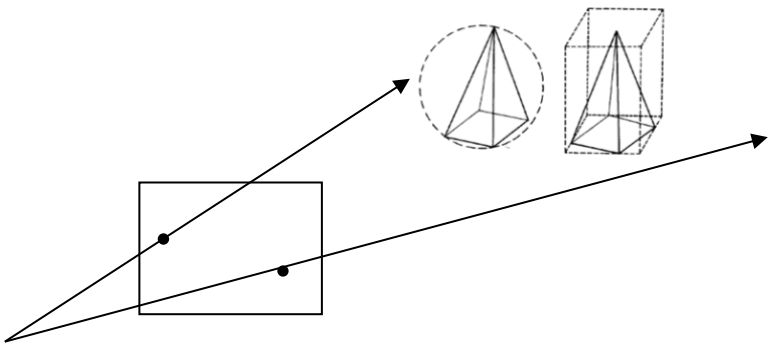


Рис.8.5. Ограничение размерности задачи поиска пересечений.

Сфера является самой простой оболочкой объекта, поскольку факт ее

пересечения определяется следующим неравенством:

$$d^2 > (x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2,$$

где (x_0, y_0, z_0) – координаты центра сферы, x, y, z – координаты точки на луче, d – радиус сферы. Если хотя бы для одной точки луча выполняется данное неравенство, объект заносится в список активных объектов.

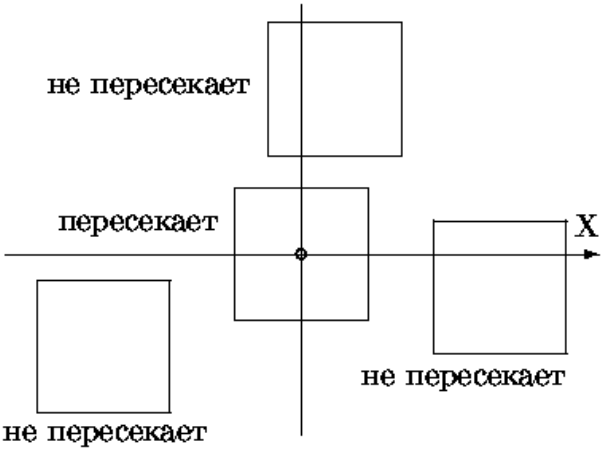


Рис.8.6. Пересечение луча с кубическими оболочками.

Найти пересечение луча с параллелепипедами или кубами несколько сложнее. Для этого используется преобразование координат, при котором центр оболочки пересекается лучом (рис. 8.6).

После того, как найдено пересечение луча с оболочкой объекта, выполняется разбиение объекта на части, которые тоже заключаются в оболочки. Аналогичным образом определяются пересечения с более мелкими оболочками, и производится дальнейшая декомпозиция объекта.

СПИСОК ЛИТЕРАТУРЫ

1. *Как выбрать ЖК монитор?* **И.А, Мухин, СПбГУТ.** 4 (292), январь 2002 г., "Компьютер-бизнес-маркет", стр. 284-291.

2. *Cramming more components inot itegrated cirquits.* **Moore, Gordon.** April 1965 г., "Electronics Magazine", стр. 4.

3. *Algorithm for computer control of a digital plotter.* **Bresenham, J.** 1, 1965 г., IBM Systems Journal, Т. 4, стр. 25-30.

4. *A clipping divider.* **Robert F. Sproul, Ivan E. Sutherland.** New York : ACM, 1968. AFIPS Joint Computer Conferences: Proceedings of the December 9-11, fall joint computer conference. стр. 765-775.

5. **О'Квин, Донни.** *Допечатная подготовка. Руководство дизайнера.* Москва : Издательский дом "Вильямс, 2003.

СПбГУ ИТМО стал победителем конкурса инновационных образовательных программ вузов России на 2007–2008 годы и успешно реализовал инновационную образовательную программу «Инновационная система подготовки специалистов нового поколения в области информационных и оптических технологий», что позволило выйти на качественно новый уровень подготовки выпускников и удовлетворять возрастающий спрос на специалистов в информационной, оптической и других высокотехнологичных отраслях науки. Реализация этой программы создала основу формирования программы дальнейшего развития вуза до 2015 года, включая внедрение современной модели образования.

КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Кафедра вычислительной техники СПбГУ ИТМО создана в 1937 году и является одной из старейших и авторитетнейших научно-педагогических школ России. Заведующими кафедрой в разное время были выдающиеся деятели науки и техники М.Ф. Маликов, С.А. Изенбек, С.А. Майоров, Г.И. Новиков. Многие поколения студентов и инженеров в Советском Союзе и за его пределами изучали вычислительную технику по учебникам «Структура ЭВМ» и «Принципы организации цифровых машин» С.А. Майорова и Г.И. Новикова, «Введение в МикроЭВМ» В.В. Кириллова и А.А. Приблуды, «Основы теории вычислительных систем» С.А. Майорова и Г.И. Новикова, А.А. Приблуды, Б.Д.Тимченко, Э.И. Махарева и др.

Основные направления учебной и научной деятельности кафедры в настоящее время включают в себя встроенные управляющие и вычислительные системы на базе микропроцессорной техники, информационные системы и базы данных, сети и телекоммуникации, моделирование вычислительных систем и процессов, обработка сигналов.

Выпускники кафедры успешно работают не только в разных регионах России, но и во многих странах мира: Австралии, Германии, США, Канаде, Германии, Индии, Китае, Монголии, Польше, Болгарии, Кубе, Израиле, Камеруне, Нигерии, Иордании и др. Выпускник, аспирант и докторант кафедры ВТ Аскар Акаев был первым президентом Киргизии.

Игорь Александрович Бессмертный

Компьютерная графика

Учебное пособие

В авторской редакции

Дизайн И.А. Бессмертный

Верстка И.А. Бессмертный

Редакционно-издательский отдел Санкт-Петербургского государственного университета информационных технологий, механики и оптики

Зав. РИО Н.Ф. Гусарова

Лицензия ИД № 00408 от 05.11.99

Подписано к печати

Заказ №

Тираж 200 экз.

Отпечатано на ризографе

Редакционно-издательский отдел

Санкт-Петербургского государственного университета информационных технологий, механики и оптики

197101, Санкт-Петербург, Кронверкский пр., 49