МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ имени академика С. П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»

М.А. КУДРИНА
К.Е. КЛИМЕНТЬЕВ

КОМПЬЮТЕРНАЯ ГРАФИКА

Рекомендовано редакционно-издательским советом федерального государственного бюджетного образовательного учреждения высшего профессионального образования «Самарский государственный аэрокосмический университет имени академика С.П. Королева (национальный исследовательский университет)» в качестве учебника для студентов, обучающихся по образовательным программам высшего профессионального образования по направлениям подготовки бакалавров «Фундаментальная информатика и информационные технологии» и «Прикладная математика и информатика»

САМАРА
Издательство СГАУ
2013

УДК 004.92(075)
ББК 32.973я7
К 888

Рецензенты: канд. физ.-мат. наук, доцент А. В. Г а в р и л о в
канд. тех. наук, доцент Е. В. С о п ч е н к о

Кудрина, М.А.
К 888 **Компьютерная графика**: учеб. / М.А. Кудрина, К.Е. Климентьев. – Самара: Изд-во Самар. гос. аэрокосм. ун-та, 2013. – 138 с.

ISBN 978-5-7883-0936-1

В учебнике рассматривается перечень вопросов, рекомендованных Государственным образовательным стандартом и соответствующих рабочей программе по курсу «Компьютерная графика».

Учебник предназначен для студентов очной формы обучения, обучающихся по направлениям подготовки: 010300 – Фундаментальная информатика и информационные технологии, 010400 – Прикладная математика и информатика. Также может быть полезен студентам заочной и очно-заочной форм обучения и, кроме того, студентам, обучающимся по смежным специальностям.

Разработан на кафедре информационных систем и технологий СГАУ.

УДК 004.92(075)
ББК 32.973я7

ISBN 978-5-7883-0936-1

ОГЛАВЛЕНИЕ

ГЛАВА 1. ВВЕДЕНИЕ В КОМПЬЮТЕРНУЮ ГРАФИКУ

Компьютерная графика (или *машинная графика*) – это дисциплина, изучающая методы генерации, преобразования, обработки и хранения моделей объектов и их изображений средствами вычислительной техники. Цель настоящего курса – ознакомление студентов с основными тенденциями в области создания и использования методов и средств работы с цифровыми изображениями.

1.1. Классификация задач компьютерной графики

В настоящее время основными направлениями компьютерной графики являются:

изобразительная компьютерная графика;
обработка изображений;
распознавание изображений;
когнитивная компьютерная графика.

Изобразительная компьютерная графика воспроизводит изображение в случае, когда исходной является информация неизобразительной природы. Примерами являются визуализация экспериментальных данных в виде графиков, гистограмм или диаграмм, вывод информации на экран компьютерных игр, синтез сцен на тренажерах (рис. 1.1). Основными задачами изобразительной компьютерной графики являются:

- построение графической модели объекта и генерация по ней изображения;
- преобразование модели и изображения.

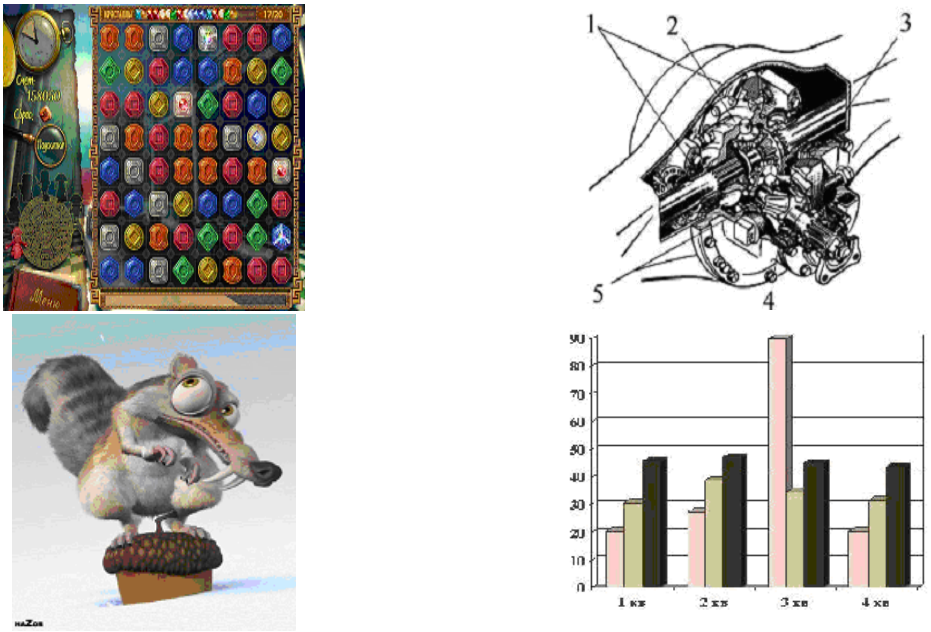
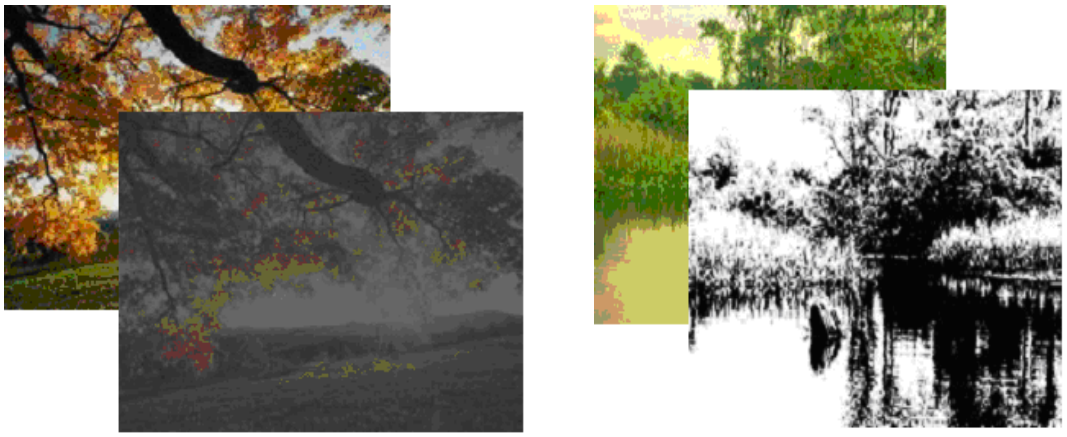


Рис. 1.1. Примеры изображений, построенных средствами изобразительной графики

Обработка изображений **(**англ. *image processing***)** рассматривает задачи, в которых и входные, и выходные данные являются изображениями. Например, это могут быть: передача изображения с устранением шумов и сжатием данных; переход от одного вида изображения к другому (от цветного к черно-белому) и т.д. – см. рис. 1.2. Целью обработки изображений может быть как улучшение его качества в зависимости от определенного критерия (реставрация, восстановление и т.п.), так и специальное преобразование, кардинально изменяющее изображение.



а) линейное контрастирование б) пороговая обработка

Рис. 1.2. Примеры обработки изображений

Распознавание изображений или *техническое зрение* **(**англ. computer vision**)** – это совокупность методов и средств, позволяющих получить формальное описание изображения и отнести его к некоторому классу. Примеры задач распознавания образов: оптическое распознавание символов, штрих-кодов, автомобильных номеров, лиц, отпечатков пальцев и т.д. (см. рис. 1.3 и 1.4). Задача распознавания является обратной по отношению к визуализации.

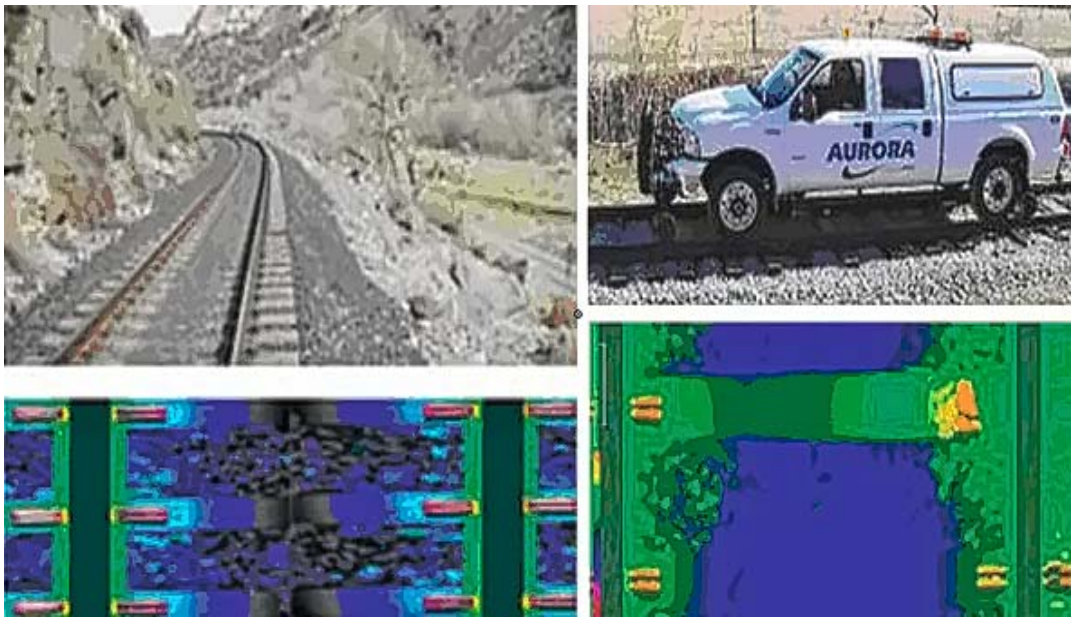
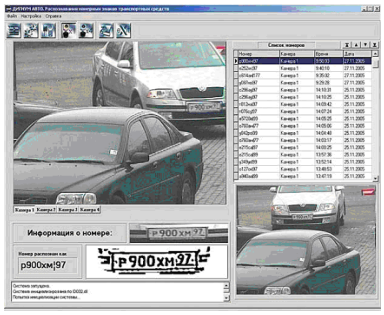
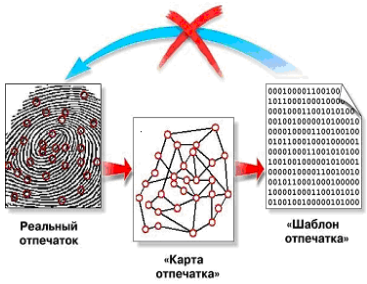


Рис. 1.3. 3*D*-камера фиксирует сломанную шпалу на скорости 50 км/ч



а) автомобильные номера

б) штрих-коды



в) отпечатки пальцев

Рис. 1.4. Примеры задач распознавания образов

Когнитивная компьютерная графика представляет собой раздел компьютерной графики, визуализирующий научные абстракции с целью рождения нового научного знания.

Примерами когнитивной графики являются фракталы. Основное свойство фракталов – самоподобие. Любой микроскопический фрагмент фрактала в той или иной степени воспроизводит его глобальную структуру. В простейшем случае часть фрактала представляет собой просто уменьшенный целый фрактал.

Берем отрезок и среднюю его треть переламываем под углом 60 градусов. Затем повторяем эту операцию с каждой из частей получившейся ломаной – и так до бесконечности. В результате мы получим простейший фрактал – триадную кривую, которую в 1904 году открыл математик Хельг фон Кох (см. рис. 1.5).

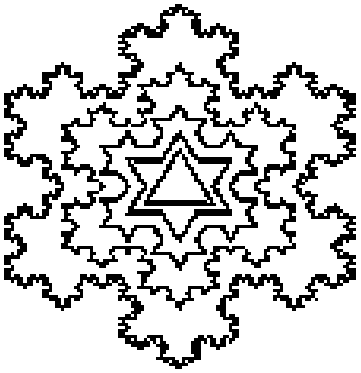
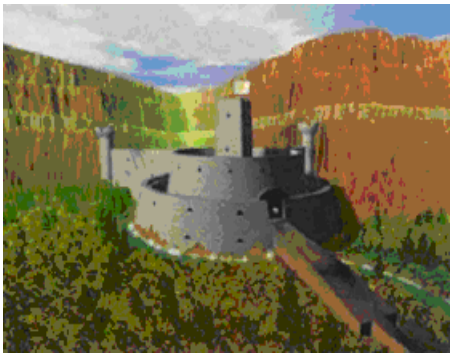


Рис. 1.5. Фрактал «снежинка Коха»

Большинство текстур местности в современных компьютерных играх представляют фракталы. Горы, лес и облака на картинке – фракталы (см. рис. 1.6).



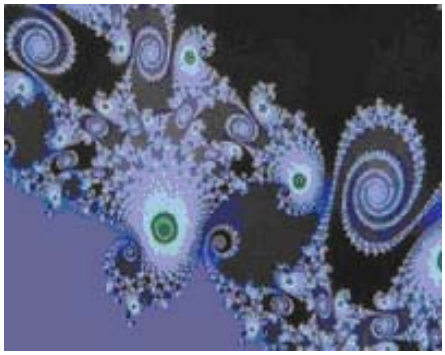
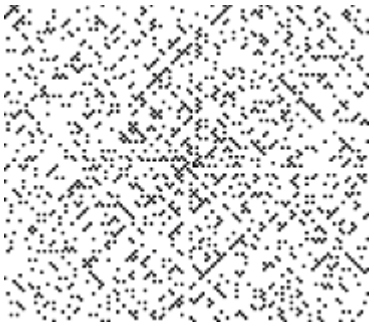


Рис. 1.6. Примеры фракталов

В качестве другого примера когнитивной графики можно привести спираль Станислава Улама (*Ulam*). Если расположить целые числа по спирали (см. рис. 1.7,а) и простые числа заменить на черные точки, а остальные – на белые, то на полученном изображении простые числа преимущественно расположатся на отрезках диагоналей (см. рис. 1.7,б).

37	36	35	34	33	32	31
38	17	16	15	14	13	30
39	18	5	4	3	12	29
40	19	6	1	2	11	28
↓	20	7	8	9	10	27
	21	22	23	24	25	26

а) нумерация точек плоскости



б) расположение простых чисел

Рис. 1.7. Иллюстрация спирали Улама

Результатом этого открытия явился новый раздел теории чисел.

1.2. Области применения компьютерной графики

Современное применение компьютерной графики очень разнообразно. Для каждого направления создается специальное программное обеспечение, которое называют *графическими программами* или *графическими пакетами*. Приведем наиболее типичные и распространенные области применения компьютерной графики.

Научная графика. Первые компьютеры использовались лишь для решения научных и производственных задач. Чтобы лучше понять полученные результаты, производили их графическую обработку, строили графики, диаграммы, чертежи рассчитанных конструкций. Первые графики на машине получали в режиме символьной печати. Затем появились специальные устройства – графопостроители (плоттеры) для вычерчивания чертежей и графиков чернильным пером на бумаге.

Конструкторская графика. Используется в работе инженеров-конструкторов, изобретателей новой техники. Этот вид компьютерной графики является обязательным элементом систем автоматизации проектирования (САПР). Графика в САПР используется для подготовки технических чертежей проектируемых устройств.

Деловая графика. Эта область компьютерной графики предназначена для создания иллюстраций, часто используемых в работе различных учреждений. Чаще всего это различные виды графиков и диаграмм.

Иллюстративная графика. Программные средства иллюстративной графики позволяют человеку использовать компьютер для произвольного рисования, черчения. Пакеты иллюстративной графики не имеют какой-то определенной производственной направленности. Поэтому они относятся к прикладному программному обеспечению общего назначения. Простейшие программные средства иллюстративной графики называются графическими редакторами.

Художественная и рекламная графика. С помощью компьютера создаются рекламные ролики, мультфильмы, компьютерные игры, видео- уроки, видеопрезентации и многое другое. Графические пакеты для этих целей требуют больших ресурсов компьютера по быстродействию и памяти. Примерами художественной графики могут служить также дизайн, верстка, цифровое фото.

Графика для Интернета. Появление глобальной сети Интернет привело к тому, что компьютерная графика стала занимать важное место в ней. Все больше совершенствуются способы передачи визуальной информации, разрабатываются более совершенные графические форматы, ощутимо желание использовать трехмерную графику, анимацию, весь спектр мультимедиа.

1.3. Основные типы изображений. Растровая и векторная графика

Цифровые изображения делятся на два больших класса: *растровые* и *векторные*.

Растровые изображения состоят из мозаики точек, которая и носит название *растр*. Изображение на экране монитора или телевизора состоит из отдельных элементов – так называемых *пикселов* (от англ. *PICture Single ELement* – элемент изображения). В отличие от геометрической точки, пиксел имеет определенную форму (квадрат, круг, шестиугольник и т.п.) и размеры.

Растровое изображение характеризуется разрешением и количеством цветов, которое может принимать каждая точка изображения. Разрешение – это количество точек на единицу длины, обычно точек на дюйм – *dpi* (англ. *dot per inch*) или пикселов на дюйм – *ppi* (англ. *pixel per inch*).

Чем большим количеством оттенков характеризуется изображение, тем большее количество двоичных разрядов требуется для их описания. Таким образом, чем качественнее изображение, тем больше размер файла.

Растровое представление обычно используют для изображений фотографического типа с большим количеством деталей или оттенков.

Наиболее распространены следующие форматы растровых изображений: BMP, PCX, JPG, GIF, TIFF, PNG и др., они подробней будут рассмотрены ниже (в разделе «Сжатие изображений»).

Основным логическим элементом *векторной* графики является геометрический объект. В качестве объекта принимаются простые геометрические фигуры (так называемые примитивы – точка, отрезок прямой или кривой линии, прямоугольник, окружность, эллипс), составные фигуры или фигуры, построенные из примитивов.

Для хранения векторного изображения в памяти достаточно сохранять лишь координаты основных точек и математическую формулу, согласно которой надо построить ту или иную фигуру. Именно поэтому векторные изображения отличаются малыми размерами файлов и их можно спокойно масштабировать, не опасаясь, что при этом потеряется качество изображения.

Наиболее распространенные форматы векторных изображений: CDR, EPS, WMF, AI и др.

Таблица 1.1. ***Сравнение характеристик растровой и векторной графики***

	Растровая графика	**Векторная графика**
Плюсы	Естественность представления (именно в таком виде изображения формируются в мозгу человека).	Масштабируются без потери качества; занимают небольшой объем памяти при хранении; корректно отображаются на любых устройствах вывода (принтеры, мониторы и т.д.).
Минусы	Большой размер файла, плохо масштабируется, при этом возможны искажения.	Практически невозможно добиться фотореалистичности.
Область применения	Сканированные снимки и снимки, полученные с помощью цифровых фотокамер; рисованные картины, насыщенные цветовыми тонами; коллажи, рисунки с применением фильтров и спецэффектов; коррекция цвета.	Плакаты и другая высококонтрастная графика; точные чертежи, архитектурные планы; традиционные логотипы и тексты с четкими и гладкими краями; текстовые документы.

1.4 Зрительный аппарат человека

Как устроен человеческий глаз? Снаружи он защищен прозрачной *роговой оболочкой* (*роговицей*), которая постоянно омывается слезами (слезной жидкостью) (см. рис. 1.8). Даже самые суровые люди за день «проливают» определенное количество слез. Глубже лежит колечко *радужной оболочки* («*радужки*»), цвет которой и имеют в виду, когда говорят о цвете глаз (см. рис. 1.9). У альбиносов в радужной оболочке нет пигмента, и поэтому она красного цвета – из-за просвечивающих кровеносных сосудов. В глазу радужка играет роль диафрагмы фотоаппарата.

Посреди радужки находится отверстие – *зрачок*. Зрачок может изменяться в диаметре от 2 до 8 мм, при этом его площадь и, соответственно, световой поток изменяются в 16 раз. Сокращение зрачка происходит за 5 секунд, а его полное расширение – за 5 минут.

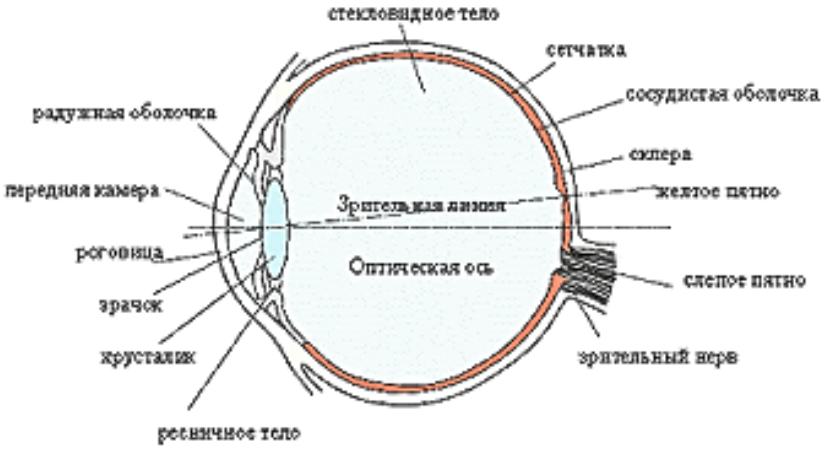


Рис. 1.8. Устройство человеческого глаза

Рис. 1.9. Радужная оболочка

Еще глубже находится *хрусталик*, который, как линза, собирает лучи в изображение на глазном дне. Астроном и физик Иоганн Кеплер в XVII веке рассмотрел устройство глаза с точки зрения оптики. Он показал, что на глазном дне формируется изображение окружающих предметов. По законам оптики такое изображение должно быть перевернутым. Именно перевернутым и видит мир новорожденный младенец. Но постепенно мозг привыкает «переворачивать» изображение обратно. Любопытно, что если надеть человеку очки, стекла которых создают перевернутое «вверх ногами» изображение, то спустя некоторое время это изображение станет восприниматься как нормальное.

Хрусталик, становясь то более выпуклым, то более плоским, может «наводить резкость» на ближние и дальние предметы. Если эта способность нарушается, возникают соответственно дальнозоркость и близорукость. Благодаря хрусталику на глазном дне формируется уменьшенное изображение окружающего нас мира. Здесь оно воспринимается сетчатой оболочкой глаза – *сетчаткой*.

Сетчатка – это многослойный лист нервной ткани, чувствительной к свету, который выстилает внутреннюю заднюю часть глазного яблока. Сетчатка расположена на пигментированной мембране, известной как *пигментированный эпителий сетчатки*, который поглощает любой свет, проходящий сквозь сетчатку. Это предотвращает обратное отражение света сквозь сетчатку и новое реагирование, что, таким образом, не разрешает зрению расплываться.

Свет проникает сквозь глаз и вызывает сложную химическую реакцию в клетках фоторецепторов сетчатки, чувствительных к свету (*палочках* и *колбочках*), что приводит к изменению функции нейрона (рис. 1.19).

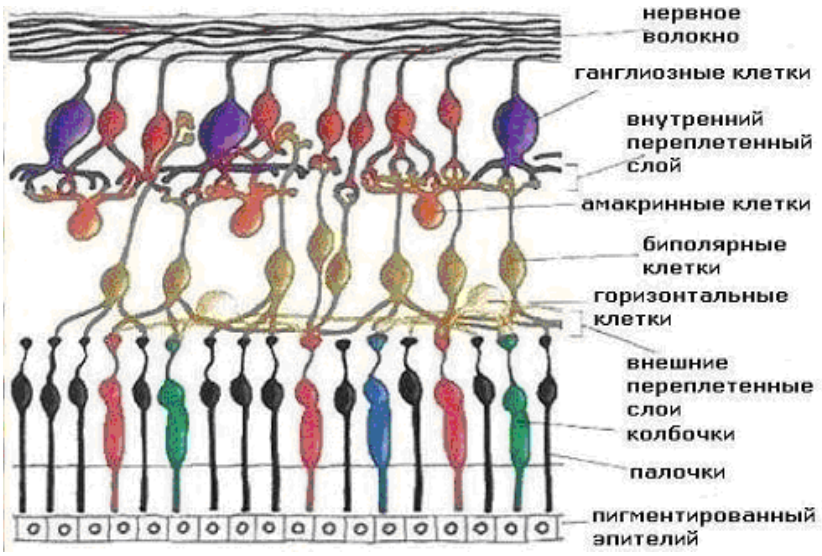


Рис. 1.10. Сетчатая оболочка глаза

Благодаря *палочкам* мы воспринимаем черно-белое изображение (ночное зрение). В глазу человека около 125 млн. палочек. В палочках находится пигмент – *родопсин*, накапливающийся в них в темноте и выцветающий на свету. Восприятие света палочками обусловлено химическими реакциями под действием света на *родопсин*.

Колбочек около 6,5 млн., их существует три типа, отличающихся светочувствительным пигментом. Колбочки обычно называют *синими*, *зелеными* и *красными*, в соответствии с наименованием цвета, к которому они наиболее чувствительны. Колбочки реагируют на свет за счет реакции *йодопсина*.

Иногда встречается *дальтонизм* – частичная цветовая слепота, один из видов нарушения цветового зрения. Эта особенность зрения впервые описана в 1794 Дж. Дальтоном, который сам страдал этим недостатком. Дальтонизм встречается примерно у 8% мужчин и у 0,5% женщин. При ослаблении восприятия одного из цветов наступает частичная цветовая слепота – *дихромазия*. Чаще всего встречается невозможность различения красного и зеленого цветов: либо зеленый воспринимается как красный (*дейтеранопия*), либо красный как зеленый (*протанопия*). Существуют специальные таблицы изображений (например, *таблицы Рабкина*), позволяющие тестировать зрительный аппарат человека на наличие или отсутствие цветовых нарушений.

Дно глаза обладает и *пигментом черного цвета*, роль которого состоит в предохранении светочувствительного аппарата от чрезмерно сильных световых раздражений. При отсутствии светового раздражения зерна этого пигмента находятся на задней поверхности сетчатки. Но при воздействии света начинается перемещение зерен навстречу падающему свету. Они проникают в слои сетчат-

ки и, поглощая значительную часть световой энергии, заслоняют палочки и колбочки от светового раздражения.

На сетчатке имеется особое место, его называют *желтым пятном*. Эта часть сетчатки имеет в середине небольшое центральное углубление – центральную ямку. По направлению к этому углублению толщина сетчатки в желтом пятне уменьшается, исчезают почти все промежуточные ее слои и остаются практически только палочки и колбочки с нервными окончаниями. В самой ямке отсутствуют и палочки, так что в ней все дно выстлано только колбочками. Диаметр желтого пятна – около 1 мм, а соответствующее ему поле зрения глаза – 6-8°. Диаметр центральной ямки – 0,4 мм, поле зрения – 1 градус.

В желтом пятне к большинству колбочек подходят отдельные волокна зрительного нерва. Вне пределов желтого пятна одно волокно зрительного нерва всегда обслуживает целые группы колбочек или палочек. По этой причине только в области ямки и желтого пятна глаз можно различать тонкие детали. В остальных местах сетчатки целые группы элементов, занимающих сравнительно большую площадь, одновременно передают раздражение одному нервному волокну, и воспринимаемая сознанием картина становится грубой, лишенной деталей.

Всякое уклонение изображения в сторону от ямки влечет за собой уменьшение четкости изображения, а когда изображение сходит с желтого пятна, то различение мелких деталей предмета совершенно прекращается. Периферическая часть сетчатки служит в основном для ориентирования в пространстве.

Распределение рецепторов на сетчатке неравномерно: в области желтого пятна преобладают колбочки, а палочек очень мало. К периферии сетчатки, наоборот, число колбочек уменьшается, и остаются одни только палочки.

Между хрусталиком и сетчаткой лежит прозрачное *стекловидное тело*, похожее на студень.

Информация от рецепторов передается в мозг по зрительному нерву, содержащему около 800 тысяч волокон. Область сетчатки, в которой волокна зрительного нерва собираются вместе и выходят из глаза, называется *слепым пятном*. В области слепого пятна нет ни колбочек, ни палочек, и этот участок сетчатки не чувствителен к свету. Диаметр слепого пятна – 1,88 мм, что соответствует полю зрения 6°. Это значит, что человек с расстояния 1 м может не увидеть предмета диаметром 10 см, если его изображение проецируется на слепое пятно.

Темновая адаптация. Происходит при переходе от больших яркостей к малым. Если в глаз первоначально попадал яркий свет, то палочки были ослеплены, родопсин выцвел, черный пигмент проник в сетчатку, заслоняя колбочки от света. Если внезапно яркость света значительно уменьшится, то вначале расширится зрачок. Затем из сетчатки начнет уходить черный пигмент, родопсин будет восстанавливаться, и, когда его наберется достаточно, начнут функциониро-

вать палочки. Так как колбочки не чувствительны к слабым яркостям, то сначала глаз не будет ничего различать, пока не начнет действие новый механизм зрения. Чувствительность глаза достигает максимального значения через 50-60 минут пребывания в темноте.

Световая адаптация. Процесс приспособления глаза при переходе от малых яркостей к большим. При этом чрезвычайно сильно происходит раздражение палочек благодаря быстрому разложению родопсина, они «ослеплены»; и даже колбочки, не защищенные еще зернами черного пигмента, раздражены слишком сильно. Только по истечении достаточного времени приспособление глаза к новым условиям заканчивается, прекращается неприятное чувство ослепления и глаз приобретает полное развитие всех зрительных функций. Световая адаптация продолжается 8-10 минут.

1.5. Принципы формирования цвета. Цветовые модели растровой графики

Назначение цветовой модели состоит в том, чтобы дать возможность удобным образом описывать цвета в пределах некоторого цветового охвата.

В компьютерной графике имеется два типа цветных объектов:

- *самосветящиеся*, излучающие объекты, такие как мониторы, плазменные панели, матрицы светодиодов и т.п.;
- *несамосветящиеся* объекты, отражающие или преломляющие падающий на них свет, такие как, например, оттиски на бумаге, светофильтры и т.п.

Для самосветящихся объектов используется «аддитивное» формирование оттенков, когда требуемый цвет формируется за счет смешения лучей света трех основных оттенков цвета. В этом случае удобно использование модели смешения RGB (от англ. *Red*, *Green*, *Blue* – красный, зеленый, синий).

Для несамосветящихся объектов используется «субтрактивное» формирование оттенков, основанное на вычитании из белого света определенных длин волн. В этом случае удобно использование модели смешения CMY (от англ. *Cyan*, *Magenta*, *Yellow* – голубой, пурпурный, желтый).

Наиболее часто в компьютерной графике используются модели RGB, CMY, YIQ, HSV и HLS.

1.5.1. Цветовая модель RGB

RGB (*Red*, *Green*, *Blue* – красный, зеленый, синий) – аппаратно-ориентированная модель, используемая в дисплеях для аддитивного формирования оттенков самосветящихся объектов (пикселов экрана). В цветовой модели, построенной на основе красного, зеленого и синего цветов, используются декартовы координаты. Система координат RGB – куб с началом отсчета (0,0,0), соответствующим черному цвету (см. рис. 1.11). Максимальное значение RGB – (1,1,1) соответствует белому цвету. Основные цвета RGB аддитивны, т.е. неко-

торый результирующий цвет можно получить путем сложения (смешения) определенных количеств каждого из основных цветов.

На главной диагонали куба, образованного равными количествами каждого из основных цветов, лежат оттенки серого цвета. Чтобы с учетом чувствительности глаза к разным цветам конвертировать RGB-изображение в градации серого (подобно тому, как показываются цветные фильмы на черно-белом экране телевизора), надо выполнить следующее преобразование:

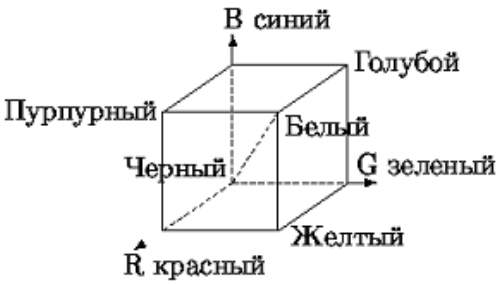


Рис. 1.11. Цветовой куб модели RGB

$$R = G = B = 0{,}299\ R + 0{,}587\ G + 0{,}114\ B.$$

1.5.2. Цветовая модель CMY

CMY (*Cyan*, *Magenta*, *Yellow* – голубой, пурпурный, желтый) – аппаратно-ориентированная модель, широко используемая в полиграфии для «субтрактивного» формирования оттенков.

Каждая из трех величин тройки (c, m, y) указывает, какое количество определенного цвета (дополнительного к соответствующему основному) подлежит исключению из белого света, чтобы получился цвет D. Т.е. если мы пишем, что $D = (c, m, y)_{CMY}$, то имеется в виду, что цвет *D* образован путем вычитания из белого цвета *c* единиц красного цвета (дополнительного к голубому), *m* единиц зеленого цвета (дополнительного к пурпурному) и *y* единиц синего цвета (дополнительного к желтому).

Отсюда непосредственно имеем соотношение между RGB и CMY системами:

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}.$$

Причем единичный вектор-строка в модели RGB – представление белого цвета, а в модели CMY – черного.

Цвета модели CMY являются *дополнительными* к цветам модели RGB, т.е. дополняющими их до белого. Таким образом, система координат CMY – тот же куб, что и для RGB, но перевернутый (см. рис. 1.12).

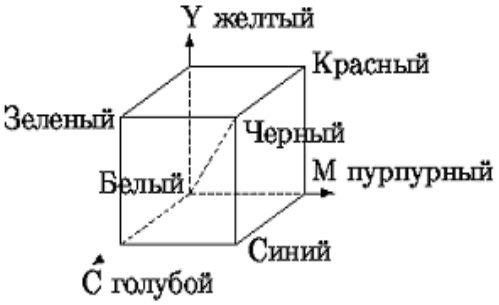


Рис. 1.12. Цветовой куб модели CMY

При освещении падающим белым светом в слое голубой краски из спектра белого цвета поглощается красная часть, затем из оставшегося света в слое пурпурной краски поглощается зеленая часть спектра, отраженный от поверхности бумаги свет еще раз подвергается поглощению, и в результате мы видим синий цвет (см. рис. 1.13).

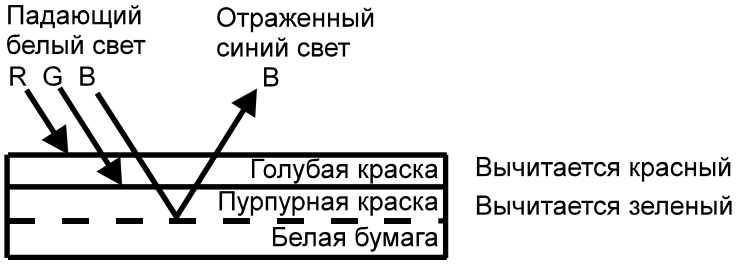


Рис. 1.13. Цвет несамосветящегося объекта

Разновидностью этой модели является модель CMYK, к которой добавлен черный цвет (от англ. *blacK* – черный).

1.5.3. Цветовая модель YIQ

Модель YIQ используется в телевизионных передающих системах, поддерживающих стандарты *M-NTSC* и *M-PAL*, и служит для сокращения передаваемой полосы частот за счет использования психофизиологических особенностей зрения. Каждый цвет в ней задается с помощью установки значений трех параметров: Y – *интенсивности* (англ. luminance) и двух цветностей I и Q, позволяющих совместно управлять созданием цвета с помощью зеленого, синего, желтого и пурпурного цветов. Так, установка минимальных значений I и Q (0, 0) приводит к получению зеленого цвета, а установка их максимальных значений (255,255) дает пурпурный цвет. Каждая из компонент YIQ модели может изменяться в диапазоне от 0 до 255. В случае использования монохромного дисплея на экране будет отображена только компонента Y.

Преобразование модели RGB в модель YIQ определяется следующим образом:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ 0{,}596 & -0{,}274 & -0{,}322 \\ 0{,}211 & -0{,}522 & 0{,}311 \end{bmatrix} \times \begin{bmatrix} R \\ G \\ B \end{bmatrix}.$$

Для обратного преобразования применяется обратная матрица

$$R = Y + 0{,}956 \cdot I + 0{,}623 \cdot Q;$$

$$G = Y - 0{,}272 \cdot I - 0{,}648 \cdot Q;$$

$$B = Y - 1{,}105 \cdot I + 1{,}705 \cdot Q.$$

Описывая цвета в модели YIQ, можно решить важную проблему, стоящую перед телевидением: два цвета, которые наши глаза воспринимают как различные, могут выглядеть совершенно одинаковыми при записи на видеопленку и последующем просмотре на черно-белом телевизионном мониторе. Этого можно избежать, если двум цветам, которые следует отличать друг от друга, присваивать различные значения яркости (величины Y) так, чтобы эти цвета изображались с различными интенсивностями.

Модель YIQ используется в стандарте JPEG.

1.5.4. Цветовая модель HSV (HSB)

Цветовые модели RGB, CMY, YIQ являются аппаратно-ориентированными. В отличие от них модель HSV ориентирована на пользователя и обеспечивает возможность явного задания требуемого оттенка цвета. HSV (*Hue*, *Saturation*, *Value) – цветовой тон* (собственно цвет), *насыщенность* (процент добавленной к цвету белой краски), *яркость* (процент добавленной черной краски). Подпространство, определяемое данной моделью, – перевернутый конус (см. рис. 1.14).

Цветовой тон H задается углом, отсчитываемым вокруг вертикальной оси. В частности, 0 – красный, 60 – желтый, 120 – зеленый, 180 – голубой, 240 – синий, 300 – пурпурный, т.е. дополнительные цвета расположены друг против друга (отличаются на 180).

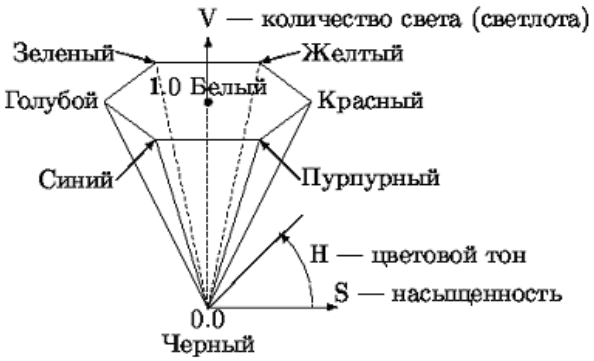


Рис. 1.14. Цветовая модель HSV

Насыщенность S – это параметр цвета, определяющий его чистоту.

Цвет с уменьшением насыщенности осветляется, как будто к нему добавляют белую краску. При значении насыщенности 0 (в центре основания конуса) любой цвет становится белым. По краю основания конуса лежат наиболее насыщенные цвета S = 1.

Яркость V – это параметр цвета, определяющий его затемненность. Уменьшение яркости цвета означает его зачернение. Яркость расположена на вертикальной оси конуса, меняется от 0 до 1. Значению V=0 соответствует вершина конуса (черный цвет), значению V=1 – основание конуса; цвета при этом наиболее интенсивны.

Промежуточные значения координаты V при S=0, т.е. на оси конуса, соответствуют серым цветам. Если S=0, то значение оттенка H считается неопределенным.

Аббревиатура HSV не самая употребительная для этой модели. В некоторых графических приложениях параметр Value («объем цвета») называется Brightness (англ. яркость), а модель соответственно HSB.

1.5.5. Цветовая модель HLS

Модель HLS (*Hue*, *Lightness*, *Saturation* – цветовой тон, яркость, насыщенность) также обеспечивает возможность явного задания требуемого оттенка цвета. Эта модель возникает в результате трансформации RGB куба в двойной конус, в котором черный цвет задается вершиной нижнего конуса и соответствует значению L = 0, белый цвет максимальной интенсивности задается вершиной верхнего конуса и соответствует значению L=1. Максимально интенсивные цветовые тона соответствуют основанию конусов с L=0,5, что не совсем удобно.

Рис. 1.15. Цветовая модель HLS

Цветовой тон H, аналогично системе HSV, задается углом поворота. Насыщенность S меняется в пределах от 0 до 1 и задается расстоянием от вертикальной оси L (перпендикулярном оси) в направлении боковой поверхности конуса. Т.е. максимально насыщенные цвета располагаются при L=0,5, S=1. В общем, систему HLS можно представить как полученную из HSV «вытягиванием» точки V=1, S=0, задающей белый цвет, вверх для образования верхнего конуса (см. рис. 1.15).

1.5.6. Цветовая гармония

Современные цветные дисплеи и принтеры позволяют получать широкий диапазон цветов. Одни цветовые сочетания хорошо гармонируют друг с другом, другие оказываются взаимно несовместимыми.

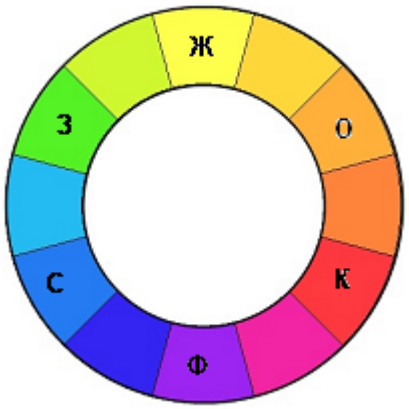


Рис. 1.16. Цветовой круг

В теории цвета цветовой круг (см. рис. 1.16) содержит в себе все цвета, видимые человеком, от фиолетового до красного. Цветовой круг показывает, как цвета связаны между собой, и позволяет определять гармоничные сочетания этих цветов. Черный, белый и серый не обозначены на цветовом круге, так как, строго говоря, они не являются цветами. Это нейтральные тона.

В цветовых схемах на рис. 1.17 приведены гармоничные сочетания цветов: 1 – диаметрально удаленная пара; 2,3 – предельно удаленные

пары; 4 – классическая триада; 5 – контрастная триада; 6 – аналогичная триада; 7, 8, 9 – четыре гармонично сочетающихся цвета. Не следует применять цвета в равных количествах. Лучше сделать один цвет фоном, а другой – акцентом на нем.

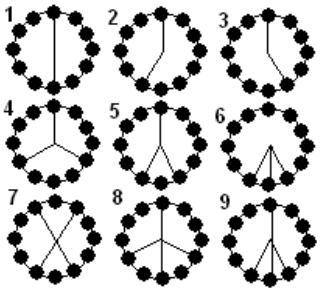


Рис. 1.17. Гармоничные сочетания цветов

1.6. Технические средства компьютерной графики

1.6.1. Устройства ввода графической информации

Для ввода графической информации используются:

- манипуляторы (мышь, трекбол, джойстик);
- сенсорная панель;
- графический планшет;
- сканер;
- цифровая фотокамера;
- цифровая видеокамера, вебкамера и пр.

Мышь – наиболее распространенный тип манипуляторов (см. рис. 1.18). Манипуляторы осуществляют непосредственный ввод информации, указывая курсором на экране дисплея команду или место ввода данных. Название «мышь» манипулятор получил в Стенфордском исследовательском институте из-за схожести сигнального провода с хвостом одноименного грызуна.


Рис. 1.18. Типичная современная мышь

Качество мыши определяется ее разрешающей способностью, которая измеряется числом точек на дюйм – dpi. Эта характеристика определяет, насколько точно курсор будет передвигаться по экрану.

Мыши различаются:

- по устройству датчика перемещения (механические, оптомеханические, оптические);
- количеству кнопок (мышь имеет от одной до трех и более кнопок);
- способу соединения (проводные и беспроводные мыши).

Беспроводные мыши используют для передачи информации инфракрасный луч или радиосигнал.

Оптомеханическая мышь скользит по плоской поверхности шариком, который передает вращение на два маленьких диска, расположенные под углом 90° друг к другу. Небольшой фотодиод освещает эти диски, а светодиод ловит от-

раженный луч, что позволяет определить угол поворота и, соответственно, перемещение мыши в том или ином направлении. Оптическая мышь тоже оснащена светодиодом, который подсвечивает поверхность, и фотодиодом, который ловит отраженный свет. Специализированный процессор, находящийся внутри мыши, анализирует отражение, выделяет отдельные участки изображения и определяет их перемещение относительно предыдущего снимка.

Трэкбол мало чем отличается от оптомеханичемкой мыши. В сущности это та же самая мышь, но перевернутая «вверх ногами», точнее вверх шаром (см. рис. 1.19). Если мышку надо возить по столу и, то в трэкболе надо просто крутить пальцами или ладонью сам шарик в разные стороны.

Рис. 1.19. Трекбол

В портативных компьютерах трэкбол нередко встраивается рядом с клавиатурой либо пристегивается с боку или спереди клавиатуры компьютера. Впрочем, и для настольных компьютеров выпускаются клавиатуры со «встроенным трэкболом». А в самых маленьких компьютерах вместо мыши и трэкбола иногда используют крошечный «тачпоинт» – небольшой цветной штырек, торчащий среди клавиш на клавиатуре, который, словно джойстик, можно отклонять в разные стороны.

Рис. 1.20. Джойстик

Джойстик – устройство ввода информации, выполненное в виде рукоятки управления, по форме напоминающей переключатель скоростей автомобиля или штурвал самолета (см. рис. 1.20). В основном джойстик используется для компьютерных игр, но иногда его применяют компьютерные художники

Рис. 1.21. Сенсорная панель на ноутбуке

Сенсорная панель – поверхность, которая может распознавать прикосновения к ней. Чаще всего сенсорные панели используются на портативных компьютерах. Сенсорные экраны устойчивы к механическим воздействиям, агрессивным средам и одновременно защищают монитор от царапин, попадания жидкостей, грязи, пыли и жира.

В настоящее время на рынке представлены три технологии сенсорных дисплеев.

1) *Резистивная.* При касании экрана происходит механическое замыкание проводящих слоев. Координаты точки касания вычисляются по изменению сопротивления, измеренного относительно углов экрана.

2) *Емкостная.* При касании экрана за счет емкости руки происходит изменение емкостных токов. Координаты точки касания вычисляются по изменению токов, измеренных относительно углов экрана.

3) *Инфракрасная.* При касании экрана происходит оптическое прерывание горизонтального и вертикального невидимых инфракрасных лучей. Решетка лучей формируется двумя линейками миниатюрных инфракрасных светодиодов. Напротив каждой их этих линеек находится линейка фотодетекторов. До недавних пор инфракрасная технология была мало распространена из-за высокой себестоимости. Однако развитие микроэлектроники привело к снижению цен, что позволило использовать эту технологию в сенсорных POS-терминалах.

Графический планшет (или *дигитайзер*, *диджитайзер*, от англ. digitizer) — это устройство для ввода рисунков от руки непосредственно в компьютер. Состоит из пера и плоского планшета, чувствительного к нажатию или близости пера (см. рис. 1.22). Также может прилагаться специальная мышь.

Рис. 1.22. Графический планшет

В современных планшетах основной рабочей частью является сеть из проводов (или печатных проводников). Эта сетка имеет достаточно большой шаг (3—6 мм), но механизм регистрации положения пера позволяет получить шаг считывания информации намного меньше шага сетки (до 100 линий на мм).

По принципу работы и технологии есть разные типы планшетов. В *электростатических* планшетах регистрируется локальное изменение электрического потенциала сетки под пером. В *электромагнитных* перо излучает электромагнитные волны, а сетка служит приемником. В обоих случаях на перо должно быть подано питание. Фирма *Wacom* (англ.) создала технологию на основе электромагнитного резонанса, которая не требует питания для пера.

Сканер (англ. *scanner*) — устройство, которое, анализируя какой-либо объект (обычно изображение, текст), создает цифровую копию изображения объекта. Процесс получения этой копии называется *сканированием*. Сканеры бывают различных конструкций: ручной, планшетный, барабанный, проекционный.

Рассмотрим принцип действия планшетных сканеров как наиболее распространенных моделей (см. рис. 1.23). Сканируемый объект кладется на стекло планшета сканируемой поверхностью вниз. Под стеклом располагается подвижная лампа, движение которой регулируется шаговым двигателем.

Свет, отраженный от объекта, через систему зеркал попадает на чувствительную матрицу (англ. CCD — *Couple-Charged Device*), далее –на АЦП и передается в компьютер. За каждый шаг двигателя сканируется полоска объекта. Все полоски потом объединяются программным обеспечением в общее изобра-

жение. Изображение всегда сканируется в формат RAW, а затем конвертируется в обычный графический формат с применением текущих настроек яркости, контрастности и т. д. Эта конвертация осуществляется либо в самом сканере, либо в компьютере — в зависимости от модели конкретного сканера. На параметры и качество RAW-данных влияют такие аппаратные настройки сканера, как время экспозиции матрицы, уровни калибровки белого и черного, и т. п. Все бытовые сканеры содержат собственные микропроцессоры, иногда это совмещенные с АЦП микропроцессоры, а иногда это микропроцессоры общего вида.

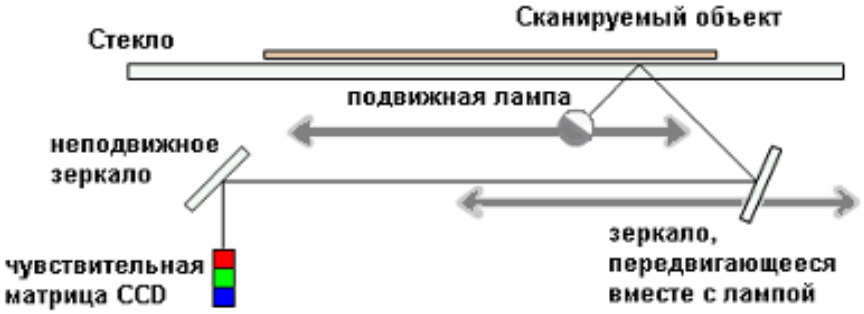


Рис. 1.23. Принцип действия сканера

Цифровая фотокамера отличается от обычного фотоаппарата тем, что изображение не фиксируется на фотопленке химическим путем, а воспринимается матрицей ПЗС («прибор с зарядовой связью»), после чего записывается в микросхемы памяти фотокамеры. Матрица ПЗС состоит из большого количества ячеек. Падающий на отдельный датчик ПЗС свет создает на нем электрический заряд, величина которого определяется интенсивностью падающего света. Изображение делится на множество ячеек, и каждая ячейка реального изображения соответствует ячейке ПЗС. Ячейки реагируют только на яркость, к цвету они безразличны, поэтому для получения цветного изображения перед матрицей ставят цветные фильтры. Каждый из пикселов регистрирует свет либо в красной, либо в зеленой, либо в синей части оптического спектра.

Рис. 1.24. Типичная цифровая камера

Основной характеристикой цифровой фотокамеры является количество пикселов матрицы ПЗС. Как правило, файлы изображения хранятся в сжатом виде в формате JPEG. Сжатие уменьшает размер файла в несколько десятков раз (см. далее). Затем изображение обрабатывается в процессоре, и на основе этих трех цветов восстанавливается вся картина.

Веб-камера (также *вебкамера*) — цифровая видео или фотокамера, способная в реальном времени фиксировать изображения, предназначенные для дальнейшей передачи по сети интернет (см. рис. 1.25).

Рис. 1.25. Веб-камера для персонального компьютера

Веб-камера содержит объектив, оптический фильтр, ПЗС или КМОП матрицу, схему цифровой обработки изображения, схему компрессии изображения и опционально веб-сервер для подключения к сети. Веб-камеры, доставляющие изображения через интернет, закачивают изображения на веб-сервер либо по запросу, либо непрерывно, либо через регулярные промежутки времени. Это достигается путем подключения камеры к компьютеру или благодаря возможностям самой камеры. Некоторые современные модели обладают аппаратным и программным обеспечением, которое позволяет камере самостоятельно работать в качестве веб-сервера, FTP-сервера, FTP-клиента и (или) отсылать изображения электронной почтой. Веб-камеры, предназначенные для видеоконференций, — это, как правило, простые модели камер, подключаемые к компьютеру, на котором запущена программа типа Instant Messenger. Модели камер, используемые в охранных целях, могут снабжаться дополнительными устройствами и функциями (такими как детекторы движения, подключение внешних датчиков и т. п.).

1.6.2. Устройства вывода графической информации

Для вывода графической информации используются следующие устройства:

- мониторы (ЭЛТ, жидкокристаллические, плазменные и т.д.);
- принтеры (матричные, лазерные, струйные, сублимационные);
- графопостроитель или плоттер;
- цифровой проектор;
- экраны на основе «электронных» чернил.

Монитор (*дисплей*) является универсальным устройством вывода информации. Классификация мониторов:

- ЭЛТ — на основе электронно-лучевой трубки (англ. *cathode ray tube, CRT*);
- ЖК — жидкокристаллические мониторы (англ. *liquid crystal display, LCD*);
- плазменный — на основе плазменной панели;
- проекционный — видеопроектор и экран, размещенные отдельно или объединенные в одном корпусе (как вариант — через зеркало или систему зеркал);

- OLED-монитор — на технологии OLED (англ. *organic light-emitting diode* — органический светоизлучающий диод);
- виртуальный ретинальный монитор — технология устройств вывода, формирующая изображение непосредственно на сетчатке глаза.

Основные параметры мониторов:

- вид экрана — квадратный или широкоформатный (прямоугольный);
- размер экрана — определяется длиной диагонали;
- разрешение — число пикселей по вертикали и горизонтали;
- глубина цвета — число отображаемых цветов (от монохромного до 32-битного);
- размер зерна или пикселя;
- частота обновления экрана;
- скорость отклика пикселей (не для всех типов мониторов).

Электронно-лучевая трубка. Источник пучка электронов – катод (см. рис. 1.26). Пучок проходит через фокусирующую систему, которая сжимает его в тонкий луч. Луч проходит через отклоняющую систему, которая изменяет его направление. Потом луч попадает на экран, покрытый светящимся веществом (люминофором). В цветных ЭЛТ используются три пучка электронов, а экран покрыт мозаикой из трех разных люминофоров. Изображение рисуется последовательно, точка за точкой. Но глаз не замечает этого, если кадр успевает нарисоваться за время $T < 1/24$ с. Современные мониторы ПЭВМ имеют частоту развертки 1/70 – 1/100 с, это более комфортно для глаза. Смена картинки для исключения мерцания производится во время обратного хода луча.

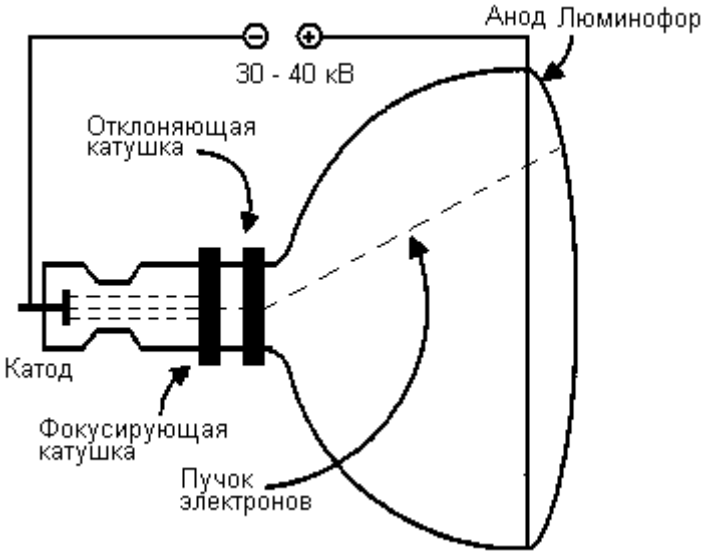


Рис. 1.26. Устройство электронно-лучевой трубки

Достоинства ЭЛТ: высокое качество цветопередачи, возможность управлять размером пиксела.

Недостатки ЭЛТ**:** электромагнитное излучение, ионизирующее излучение, мерцание, высокое напряжение, ядовитые вещества.

Жидкокристаллический экран. Работа ж/к экранов основана на таком свойстве света, как поляризация. Обычный свет является неполяризованным, т.е. амплитуды его волн лежат во множестве плоскостей. Однако существуют вещества, способные пропускать свет только в одной плоскости (поляризаторы). Если взять два поляризатора, плоскости поляризации которых расположены под углом 90º друг к другу, свет через них пройти не сможет. Если же расположить между ними что-то, что сможет повернуть вектор поляризации света на нужный угол, мы получим возможность управлять яркостью свечения.

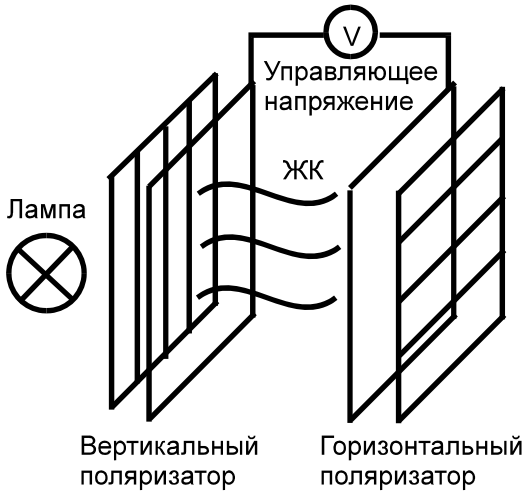


Рис. 1.27. Ячейка ЖК-матрицы

В упрощенном виде матрица ж/к дисплея состоит из следующих частей (см. рис. 1.27):

- ртутная лампа подсветки;
- система отражателей и полимерных световодов, обеспечивающая равномерную подсветку;
- фильтр-поляризатор;
- стеклянная пластина-подложка с контактами;
- жидкие кристаллы;
- еще один поляризатор;
- снова стеклянная подложка с контактами;
- в цветных матрицах добавляется еще и цветной фильтр.

В широком смысле матрицы делятся на «пассивные» и «активные». В *пассивных матрицах* управление производится попиксельно, т.е. по порядку от ячейки к ячейке в строке. Используются для небольших размеров диагоналей и невысокой плотности отображения. *Активные матрицы* (технология TFT – Thin Film Transistor тонкопленочный транзистор). Благодаря TFT появилась возможность управлять каждым пикселом отдельно. Транзисторы изолированы друг от

друга и подведены к каждой ячейке матрицы. Усилитель TFT для каждого субпиксела применяется для повышения быстродействия, контрастности и четкости изображения дисплея.

Такие ЖК-мониторы лишены основных недостатков ЭЛТ: мерцания изображения, плохой четкости картинки из-за недостаточной фокусировки луча и несведения; нет и самого вредного – статического потенциала экрана.

Недостатки**:** низкое качество в видеорежимах, требующих иного количества пикселов; низкая яркость изображения; малая скорость обновления изображений; узкий угол, под которым видно изображение.

Плазменная панель (*газоразрядный экран*) использует в своей работе явления электрического разряда в газе и возбуждаемого им свечения люминофора. В качестве газовой среды обычно используется неон или ксенон. Экран состоит из большого количества маленьких лампочек «дневного света».

Достоинства: как у ж/к экранов. Недостатки: возможность применения только на больших плоскостях (в «плазменных панелях»); невысокий ресурс, т.к. электроды на лампочках со временем «выгорают».

Принтер (англ. *printer* — печатник) — устройство печати цифровой информации на твердый носитель, обычно на бумагу. Процесс печати называется вывод на печать, а получившийся документ — распечатка или твердая копия. Принтеры бывают *струйные, лазерные, матричные* и *сублимационные*, а по цвету печати — *черно-белые* (монохромные) и *цветные*. Иногда из лазерных принтеров выделяют в отдельный вид «*светодиодные*» принтеры.

Матричные принтеры – это принтеры ударного действия. Изображение формируется печатающей головкой, которая состоит из набора иголок (игольчатая матрица), приводимых в действие электромагнитами. Головка передвигается построчно вдоль листа, при этом иголки ударяют по бумаге через красящую ленту, формируя точечное изображение. Основное распространение получили 9- и 24-игольчатые принтеры. Принтеры с 24 иголками называют *LQ* (англ. *Letter Quality* — качество пишущей машинки). Разновидностью матричных являются *термопринтеры*, которые не используют красящей ленты. Вместо этого применяется специальная теплочувствительная бумага, которая темнеет в местах прикосновения нагретых игл.

Матричные принтеры, несмотря на то что многие считают их устаревшими, все еще активно используются в тех случаях, когда необходимо получить второй экземпляр документа через копирку. Матричные принтеры применяются также в банках, так как они обеспечивают защиту документов от подделок, оставляя на них не только изображения, но и их отпечатки. Расходные материалы для матричных принтеров самые дешевые.

Недостатки матричных принтеров состоят в том, что они печатают медленно, производят много шума и качество печати невысоко.

Струйные принтеры используют чернильные печатающие головки, которые под давлением выбрасывают на бумагу из ряда мельчайших отверстий капельки чернил различных цветов (см. рис. 1.28). Перемещаясь вдоль бумаги, печатающая головка оставляет строку символов и изображения.

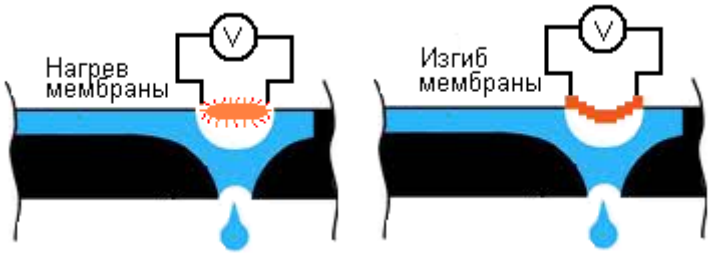


Рис. 1.28. Принципы действия струйных принтеров

К достоинствам струйных принтеров следует отнести возможность создания высококачественного цветного изображения. К недостаткам: специальные требования к чернилам и бумаге, сложность в обслуживании.

Лазерные принтеры (см. рис. 1.29) работают следующим образом. Сначала по поверхности фотобарабана равномерно распределяется статический заряд, после этого лазерным лучом (либо излучением от светодиодной линейки) на фотобарабане снимается заряд, – тем самым на поверхность барабана помещается скрытое изображение. Далее на фотобарабан наносится тонер. Тонер притягивается к разряженным участкам поверхности фотобарабана, сохранившей скрытое изображение. После этого фотобарабан прокатывается по бумаге, и тонер переносится на бумагу валом переноса. После этого бумага проходит через блок термозакрепления для фиксации тонера, а фотобарабан очищается от остатков тонера и разряжается в узле очистки.

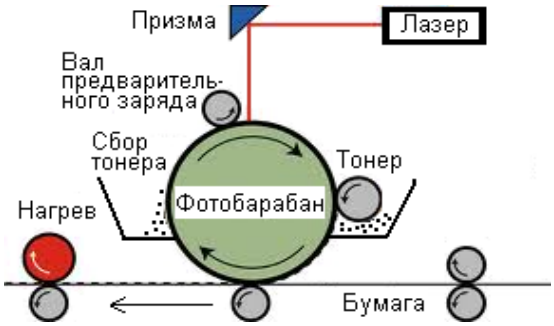


Рис. 1.29 Принцип действия лазерного принтера

Достоинства лазерных принтеров: высокая разрешающая способность, высокая скорость печати. Недостатки**:** высокая стоимость расходных материалов, нестойкость изображения (можно «стереть» в микроволновой печи).

Сублимационные принтеры работают на основе принципа «сублимации» — быстрого нагрева красителя, когда минуется жидкая фаза и сразу образуется пар. Чем меньше порция, тем больше фотографическая широта (динамический диапазон) цветопередачи. Пигмент каждого из основных цветов, а их может

быть три или четыре, находится на отдельной (или на общей многослойной) тонкой лавсановой ленте. Печать окончательного цвета происходит в несколько проходов: каждая лента последовательно протягивается под плотно прижатой термоголовкой, состоящей из множества термоэлементов. Эти последние, нагреваясь, возгоняют краситель. Точки, благодаря малому расстоянию между головкой и носителем, стабильно позиционируются и получаются весьма малого размера.

К серьезным проблемам сублимационной печати можно отнести разрушаемость применяемых чернил под воздействием ультрафиолета. Также невысока скорость печати.

Графопостроитель (от греч. γράφω — пишу, рисую), он же *плоттер* — устройство для автоматического вычерчивания с большой точностью рисунков, схем, сложных чертежей, карт и другой графической информации на бумаге размером до А0 или кальке (см. рис. 1.30).

Рис. 1.30. Плоттер

Типы графопостроителей:

- рулонные и планшетные;
- перьевые, струйные и электростатические;
- векторные и растровые.

Назначение графопостроителей – высококачественное документирование чертежно-графической информации.

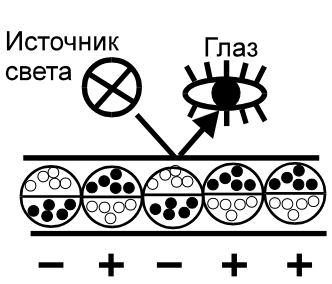
Графопостроители можно классифицировать следующим образом:

- по способу формирования чертежа — с произвольным сканированием и растровые;
- по способу перемещения носителя — планшетные, барабанные и смешанные (фрикционные, с абразивной головкой);
- по используемому инструменту (типу чертежной головки) — перьевые, фотопостроители, со скрайбирующей головкой, с фрезерной головкой.

Также плоттерами называют широкоформатные принтеры. Это не совсем корректно, однако де-факто уже является стандартом.

Экраны на основе *электронных чернил* (англ. E-inc) применяются в устройствах чтения электронных книг (см. рис. 1.31). Принцип действия основан на изменении пространственной ориентации крохотных прозрачных капсул, одна из половинок которых заполнена светлым красителем, а другая – темным.

Типичные размеры экрана 800×600 пикселов. Энергия тратится только в момент изменения ориентации капсул, поэтому устройства на основе электронных чернил очень экономны. Главным достоинством таких экранов является их яркость, в точности соответствующая яркости отраженного света, то есть с точки зрения оптических свойств они полностью соответствуют бумаге.



а) принцип действия

б) типичная «электронная книга»

Рис. 1.31. Устройства на основе электронных чернил

Рис. 1.32. Проектор

Мультимедийный проектор (также используется термин «*цифровой проектор*») — с появлением и развитием цифровых технологий это наименование получили два, вообще говоря, различных класса устройств, различающихся принципом действия и назначением:

- на вход устройства подается видеосигнал в реальном времени (аналоговый или цифровой), и устройство проецирует изображение на экран (см. рис. 1.32);
- устройство получает на отдельном или встроенном в устройство носителе или из локальной сети файл или совокупность файлов (слайдшоу) — массив цифровой информации, декодирует его и проецирует видеоизображение на экран, возможно, воспроизводя при этом и звук.

1.7. Видеоподсистема ПЭВМ

Видеоподсистема ПЭВМ состоит из двух взаимодействующих устройств:

- видеоконтроллера (синонимом этого термина является термин «видеоадаптер», а если устройство реализовано на отдельной плате, то еще «видеокарта» и «видеоплата»);
- дисплея.

Практически все видеоконтроллеры в прошлом и настоящем разрабатываются и реализуются в предположении, что устройством отображения является электронно-лучевой монитор. Поэтому для совместимости с ЖК-мониторами последние содержат специальные интерфейсные схемы преобразования управляющих и информационных сигналов.

Изображение на электронно-лучевом экране формируется в результате построчного перемещения по люминофору экрана световой точки.

Типичное количество строк в телевизоре составляет 625, а на компьютерных дисплеях – от 200 до 2048. При прорисовке одного кадра на «прямой ход» луча затрачивается 80-95% общего времени (см. рис. 1.33).

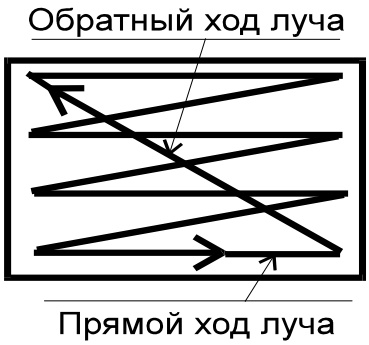


Рис. 1.33. Перемещение луча по экрану

Видеоконтроллер состоит из следующих функциональных блоков (см. рис. 1.34).

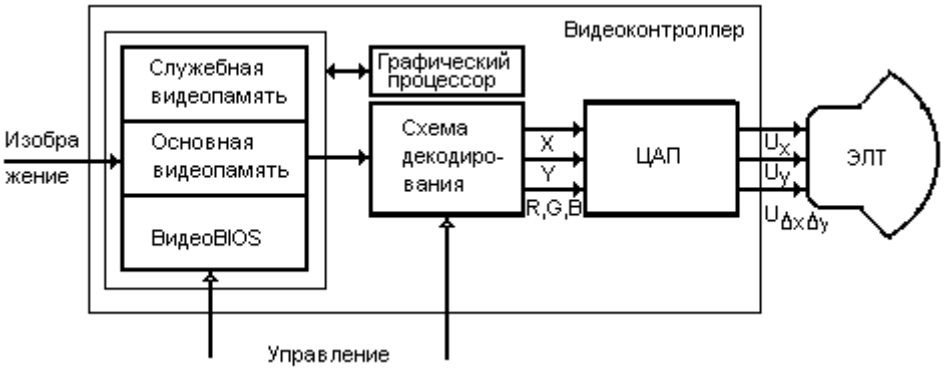


Рис. 1.34. Функциональная схема видеоподсистемы ПЭВМ

Видеопамять. Это набор ячеек памяти, физически размещающихся на плате видеоконтроллера. Он делится на три части:

1) *основная* видеопамять;

2) *служебная* видеопамять;

3) видео-BIOS.

В технических характеристиках видеоконтроллера обычно указывается суммарный объем видеопамяти, например – 640 Мб.

В *основной памяти* формируется один или несколько «образов» изображения. В общем случае (это зависит от видеорежима работы контроллера) основная память логически разделяется на несколько страниц и слоев (см. рис 1.34). Одна из страниц может быть назначена активной (текущей), любое изменение информации на ней сразу же отображается на экране. Нулевой слой видеопамяти является частью адресного пространства ПЭВМ в районе адресов A0000h÷BFFFFh.

Служебная память используется для внутренних операций видеоконтроллера, например, для хранения временных данных при расчете изображений. *Видео-BIOS* – защищенный от записи фрагмент памяти с адресами 0C0000h÷0EFFFFh (см. табл. 1.2), который содержит стандартные процедуры для управления конкретной моделью видеоконтроллера. В стандартном режиме

работы процессора они программно доступны через прерывание 10h. Кроме того, в современных видеоконтроллерах часть видео-BIOS содержит «скрытый» код, предназначенный не для выполнения процессором ПЭВМ, а для выполнения *графическим процессором*.

Схема декодирования конвертирует информацию об изображении в числа, пропорциональные управляющим напряжениям.

Цифро-аналоговый преобразователь (ЦАП) конвертирует числа в управляющие напряжения для электронно-лучевой трубки.

Графический процессор – специализированное арифметико-логическое устройство со специфической системой команд, ориентированной на целочисленные арифметические операции, действия над векторами и матрицами. Основное назначение графического процессора – выполнение расчетных операций при построении изображений, но в последние годы эти процессоры активно используются для наращивания вычислительной мощности самого компьютера.

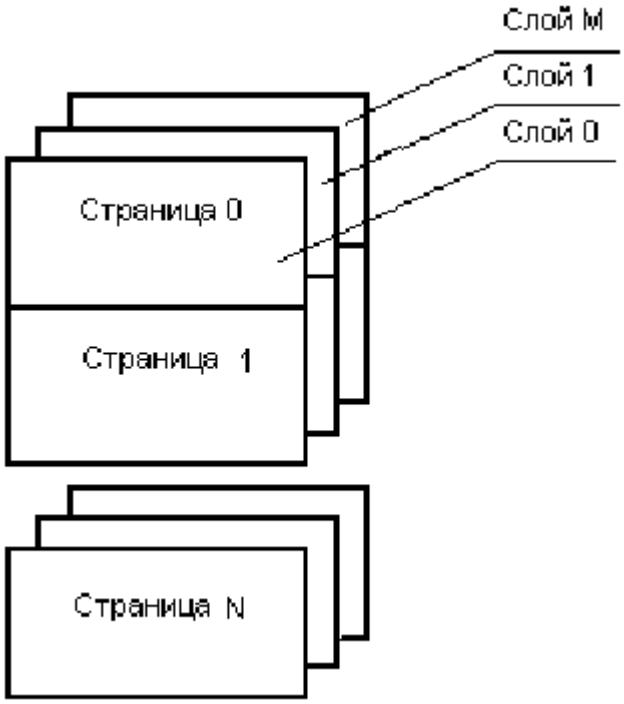


Рис. 1.34. Общая структура основной видеопамяти

Исторически, совместно с ПЭВМ использовалось большое количество видеоконтроллеров. Исторические типы контроллеров приведены в табл. 1.2. Более поздние модели видеоконтроллеров, как правило, поддерживают режимы работы всех более ранних моделей.

Т а б л и ц а 1.2. ***Исторические типы видеоконтроллеров***

Год создания	**Модель**	**Предельные изобразительные возможности**
1981 г.	MDA	2-цветные алфавитно-цифровые изображения
1982 г.	CGA	4-цветные графические изображения 320×200 точек
1984 г.	EGA	16-цветные графические изображения 480×350 точек
1987 г.	VGA	256-цветные графические изображения 360×480 точек
1991 г. и позже	MCGA, SVGA, XGA, PGA	24- и 32-битные графические изображения, до 2048 точек по горизонтали и вертикали
1996 и позже	Модель 3D Voodoo и ее развитие	24- и 32-битные графические изображения, до 2048 точек по горизонтали и вертикали, расчет изображений без участия центрального процессора

В настоящее время используются сотни моделей различных производителей. В современных видеоконтроллерах схема декодирования и ЦАП реализованы на единой микросхеме, которая носит наименование *видеопроцессора*. Видеопроцессор работает на тактовых частотах 100-300 МГц и выше. Современный видеопроцессор способен не только осуществлять декодирование и цифро-аналоговое преобразование данных, извлекаемых из видеопамяти, но и выполнять многочисленные микропрограммы, реализующие полный цикл расчета трехмерных изображений (эти микропрограммы называются *шейдерами* и располагаются в «скрытой» части видео-BIOS).

Видеоконтроллер может быть реализован в виде отдельного устройства либо интегрирован в чипсет материнской платы. Во втором случае в качестве видеопамяти может быть использована часть оперативной памяти ПЭВМ. Если же видеоконтроллер представляет собой отдельную плату, то возникает необходимость обеспечить быстродействующий интерфейс между центральным процессором ПЭВМ и видеопроцессором. Пока видеопроцессоры не обладали расширенными возможностями, с задачей передачи данных вполне справлялись системные шины ISA (пропускная способность 5,55 Мбит/с) и EISA (32 Мбит/с). По мере роста быстродействия центрального процессора и усложнения задач, возлагаемых на видеопроцессор, ужесточались и требования к интерфейсу. В 1990-х годах использовались шины VLB и PCI (132 Мбит/с). В настоящее время основным интерфейсом для подключения видеоплат являются различные модификации шины AGP (528 Мбит/с и выше) и PCI-Express (до 2,5 Гбит/с и выше).

Ситуация на рынке видеоконтроллеров быстро меняется: растут функциональные возможности и тактовые частоты видеопроцессоров, объемы видеопамяти. В настоящее время законодателями мод на рынке видеопроцессоров являются компании Intel, nVidia, ATi, Matrox и др. Видеочипы и видеоконтроллеры этих производителей используются не только в ПЭВМ, но и в мощных графических станциях, в телевизорах, DVD-проигрывателях, игровых приставках и прочих цифровых устройствах.

1.7.1. Обзор видеорежимов

Видеорежим – это стандартизованный набор установок видеоконтроллера, позволяющий формировать изображение с определенными характеристиками. Обычно описывается как X×Y×C, где X – максимальное количество знаков или точек по горизонтали; Y – максимальное количество знаков или точек по вертикали; C – максимальное количество цветов.

Алфавитно-цифровые режимы 0 и 1 с разрешением 40×25×16.

При использовании видеоадаптеров EGA или VGA не существует функциональных различий между режимом 0 и режимом 1. В данных режимах дисплей отображает цветную текстовую (алфавитно-цифровую) информацию – 25 строк и 40 столбцов.

Для отображения каждого символа используется матрица 8 на 8 пикселов, что соответствует низкому качеству изображения (можно различить отдельные пиксели, из которых состоит символ).

Символы текста можно отображать в 8 основных и 8 дополнительных цветах. Последние имеют большую интенсивность, чем основные. Для каждого символа можно независимо задать его цвет и цвет фона. Список стандартных и дополнительных цветов представлен в табл. 3.

Т а б л и ц а 1.3. ***Стандартные и дополнительные цвета***

Стандартный цвет	**Дополнительный цвет**
черный	серый
синий	светло-синий
зеленый	светло-зеленый
морской волны	голубой
красный	светло-красный
фиолетовый	малиновый
коричневый	желтый
белый	ярко-белый

Для видеоадаптеров EGA и VGA можно изменить используемую палитру цветов. EGA с улучшенным цветным дисплеем позволяет выбрать 16 цветов из 64 возможных, а VGA-16 из 262144.

В режимах 0 и 1 адаптеры EGA и VGA поддерживают восемь страниц видеопамяти. Страницей называется часть видеопамяти, полностью определяющая содержимое одного экрана дисплея. Одна из этих восьми страниц является активной, то есть ее содержимое отображается на экране. Для изменения активной страницы можно либо вызвать соответствующую функцию BIOS, либо непосредственно изменить содержимое регистра начального адреса, расположенного в контроллере электронно-лучевой трубки.

Алфавитно-цифровые режимы 2 и 3 с разрешением 80×25×16. Для видеоадаптеров EGA и VGA данные режимы не имеют различий. Это стандартные режимы для MS-DOS.

Видеопамять имеет 1 слой и 8 страниц. По умолчанию активна страница 0, ее стартовый адрес B800h:0. Один знак на экране кодируется двумя соседними байтами:

- четный (0, 2, 4…) байт содержит числовой код знака;
- нечетный (1, 3, 5…) байт содержит описание цвета знака и цвета фона.

Формат байта атрибутов:

7	6	5	4	3	2	1	0
B3	B2	B1	B0	S3	S2	S1	S0

Биты S0-S3 кодируют один из 16 цветов знака (0 – черный, 1 – синий,… 7 – светло-серый, 8 – темно-серый, 9 – голубой, … 15 – белый). Биты B0-B2 кодируют один из 8 цветов фона. Бит B3 либо является дополнительным битом цвета фона (тогда фон может выбираться не из 8, а из 16 цветов), либо кодирует признак мерцания знака (1-мерцает, 0-нет). Режим бита B3 переключается с цвета на мерцание функцией видеоBIOS с номером 1003h:

```
r.ax:=$1003;
r.bl:= 0 – цвет; 1-мигание.
Intr($10, r);
```

На самом деле изображение знака аппаратно (при помощи *знакогенератора*) моделируется матрицей светящихся точек: на CGA: 8×8, на EGA – 8×14, на VGA – 8×16 или 9×16. ВидеоBIOS хранит внутри таблицы с описаниями формы всех знаков (без русских букв!). Эти таблицы могут быть перепрограммированы.

Графические видеорежимы 4 и 5 с разрешением 320×200×4. При отображении могут использоваться либо четыре основных, либо четыре альтернативных цвета: 1) черный, зеленый, красный, желтый; 2) черный, голубой, малиновый, белый. Четверка цветов может выбираться при помощи функции 0Bh видеоBIOS. Ниже приведен фрагмент кода, позволяющий изменить четверку цветов:

```
r.ah:=$0B;
r.bh:=1;
r.bl:= номер цветового набора, 0 или 1
Intr($10, r);
```

Видеопамять в режиме 4 имеет 1 слой и 1 страницу, разделенную на две части (см. рис. 1.35). Первая часть начинается по адресу 0B800h:0 и содержит описания четных (0, 2, 4…) строк, вторая – с адреса 0B800:2000h и содержит описания нечетных (1, 3, 5…) строк изображения. Цвет каждой точки кодируется двумя соседними битами.

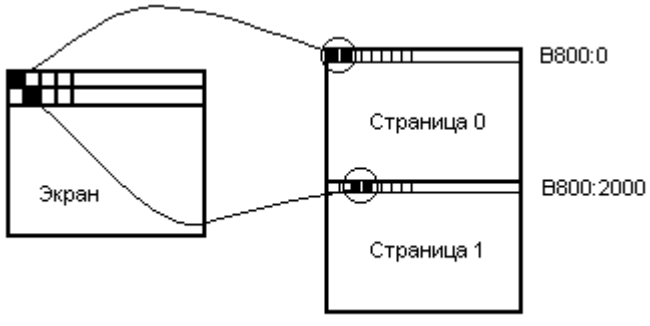


Рис. 1.35. Видеопамять в режиме 4

Формулы для расчета адреса пиксела в видеопамяти:

- Nбайта = 50h × ((y - 1)/2) + (x/4) – для четных строк;
- Nбайта = Nбайта + 2000h – для нечетных строк;
- Nбита = 7 – (x mod 4) × 2.

Графический режим 6 с разрешением 640×200×2. Видеопамять начинается с адреса B800h:0, имеет 1 слой и 1 страницу, разделенную на две части для четных и нечетных строк (см. режим 4). Два цвета (черный или белый) кодируются одним битом.

Формулы для расчета:

- Nбайта = 50h× (y/2) + (x/8) – для четных строк;
- Nбайта = Nбайта + 2000h – для нечетных строк;
- Nбита = 7 - (x mod 8).

Графический режим 12h с разрешением 640×480×16 – режим, который устанавливается в MS Windows по умолчанию. Видеопамять начинается с адреса A000h:0, имеет одну страницу и 4 слоя. Цвет кодируется 4 битами, расположенными под одним и тем же номером в одном и том же байте, но в разных слоях (см. рис. 1.36).

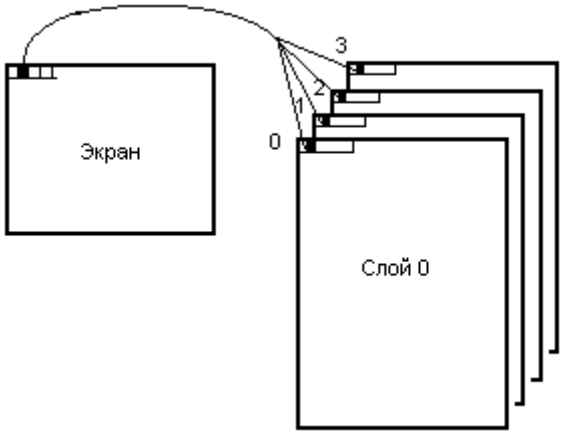


Рис. 1.36. Видеопамять в режиме 12h

Формулы для расчета:

- Nбайта = 50h × y + x/8;
- Nбита = 7 - (x mod 8).

Для того чтобы поставить точку на экране, необходимо:

1) прочитать 4 байта из 4 разных слоев;
2) исправить нужные биты;
3) записать 4 байта на прежние места.

Внимание! При этом используются внутренние регистры контроллера.

Графический режим 0Fh с разрешением 640×350×4. Видеопамять начинается с адреса A000h:0, имеет одну страницу и 2 слоя. Цвет кодируется 2-мя битами, расположенными под одним и тем же номером в одном и том же байте, но в разных слоях (см. режим 12h).

Формулы для расчета:

- Nбайта = 50h × y + x/8;
- Nбита = 7 - (x mod 8).

Графический режим 13h с разрешением 320×200×256. Видеопамять начинается с адреса A000h:0, имеет 1 слой и 1 видеостраницу. Одной точке изображения соответствует целиком байт видеопамяти, цвет в диапазоне 0-256 задается всеми восемью битами.

Формула для расчета:

- Nбайта = 140h × y + x.

Пример:

```
{Отобразить пиксел с координатами (123;45) и заданным цветом}
x: = 123; (* Координата X *)
y: = 45;  (* Координата Y *)
Mem[$140*y + x]:=цвет;
```

Графический режим с разрешением 320×400×256. Видеоадаптер VGA технически способен работать в этом видеорежиме, но он не является стандартным и не поддерживается процедурами видеоBIOS.

1.7.2. Программирование видеоконтроллеров

Видеоконтроллеры можно программировать:

- средствами видеоBIOS;
- непосредственно через регистры контроллера.

Программирование средствами видеоBIOS. ВидеоBIOS предоставляет десятки сервисных функций для программирования видеоконтроллера в стандартных видеорежимах (с 1 по 13h). Функции доступны через прерывание с номером 10h, поэтому использовать их можно только в MS-DOS программах.

Недостатки:

- это очень медленно работающие функции;
- невозможно использование нестандартных возможностей видеоконтроллера;
- часть функций стандартизована, а часть может просто отсутствовать (например, VESA-функции).

Поэтому функции видеоBIOS практически никогда не используются для создания компьютерных игр, демонстрационных программ и других графических приложений.

1. Установка видеорежима.

```
r.ah:=0;
r.al:=  номер режима
Intr($10, r);
```

2. Определение текущего видеорежима.

```
r.ah:=$F;
Intr($10,r);
Writeln('Видеорежим №', r.al);
```

3. Рисование точки.

```
r.ah:=$Ch;
r.al:=цвет
r.bh:=номер страницы видеопамяти (обычно 0)
r.cx:=X;
r.dx:=Y;
Intr($10,r);
```

4. Определение цвета точки.

```
r.ah:=$D;
r.bh:=номер страницы видеопамяти
r.cx:=X;
r.cy:=Y;
Intr($10, r);
Writeln('Цвет точки=', r.al);
```

5. Получение информации о таблицах алфавитно-цифровых знаков (символов).

Эти шрифты используются в стандартных текстовых и графических режимах (не в Windows). Изображения знаков хранятся поточечно, каждой точке соответствует один бит (см. рис. 1.37).

Рис. 1.37. Кодирование знаков 8x8

```
r.ax:=$1130;
r.bh:= Вид запроса
Intr($10, r);
Writeln('Высота знака в точках=', r.cl);
Writeln('Умещается на экране строк=', r.dl+1);
Writeln('Адрес в памяти для таблицы описания знаков  =',
r.es, ':', r.bp );
```

Вид запроса кодируется в соответствии с табл. 1.4.

Т а б л и ц а 1.4. ***Кодирование запросов***

Код запроса	**Какая информация запрашивается**
2	Информация про шрифт 8x14
3	Информация про шрифт 8x8 (1-я половина)
4	Информация про шрифт 8x8 (2-я половина)
5	Информация про шрифт 9x14
6	Информация про шрифт 8x16
7	Информация про шрифт 9x16

6. Установка своей таблицы знаков. Можно поменять стандартные изображения знаков на свои (так поступают программы-русификаторы в MS-DOS, поскольку по умолчанию в видеоBIOS нет описания изображений русских букв).

```
r.ax:=$1100;
r.es:=Seg(Table); адрес таблицы с новым набором
r.bp:=Ofs(Table); символов
r.cx:= число загружаемых символов (1-256)
r.dx:= относительное смещение первого изменяемогосимвола (0-255)
r.bl:= номер загружаемой таблицы знакогенератора для EGA 0-3, для VGA 0-7;
r.bh:= число байт отводимых на символ в таблице символов (1-32);
```

Низкоуровневое программирование видеоконтроллера. Видеоконтроллер содержит несколько сотен регистров, разделенных на группы. Регистры программно доступны через порты ввода/вывода при помощи машинных команд IN и OUT. С точки зрения доступа имеется два больших класса регистров:

- внешние регистры;
- внутренние регистры.

Внешние регистры связаны с уникальными номерами портов, например регистр состояния ввода 1 всегда доступен через порт 3DAh.

Внутренние регистры по назначению и использованию разбиты на группы. С каждой группой связаны всего два порта: индексный порт и порт данных. В *индексном порту* задается номер интересующего регистра (0, 1, 2 и т.д.). Через *порт данных* ведется обмен информацией с указанным регистром.

Регистр состояния ввода 1 (внешний регистр). Адрес порта ввода-вывода 3DAh.

7	6	5	4	3	2	1	0
				X			

Если значение бита 3 равно 0, то идет прямой ход луча; если 1 – то обратный ход. Если изменять изображение во время прямого хода луча, то возможно подергивание и искажение экрана, поэтому в компьютерных играх запись в видеопамять обычно выполняется во время обратного хода.

Внутренние регистры ЦАП позволяют устанавливать и изменять нумерацию отображаемых цветов:

- индексный регистр чтения палитры – порт 3C7h.
- индексный регистр записи палитры – порт 3C8h.
- регистр данных для доступа к палитре – порт 3C9h.

В видеоконтроллере хранится *палитра* (или *таблица цветов*) – таблица соответствия номера цвета и конкретного значения. При включении компьютера эта таблица инициализируется стандартными значениями: 0 – черный, 1 – синий, 2 – зеленый и т.д. Эта таблица цветов (палитра) может быть перепрограммирована.

Для режима 13h (320×200×256) палитра представляет собой массив из 256 строк (по количеству цветов), каждая из строк состоит из 3 байтов, каждый из этих байтов содержит компоненту R, G или B (см. рис. 1.38). Реально в каждом байте используются 6 битов, т.е. каждая составляющая может принимать значения от 0 до 63.

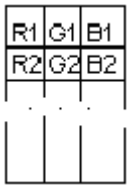


Рис. 1.38. Структура 256-цветной палитры

Таким образом, всего возможно $64\times64\times64=2^{18}=262144$ различных цветов, из которых на экране одновременно может присутствовать только 256.

Как прочитать текущую палитру:

```
Type RGB = record R, G, B : byte end;
Var Pal : array [0..255] of RGB;
…
Port[$3C7]:=i;
{Три подряд чтения из одного порта дают доступ к разным регистрам
После тройного чтения индексный регистр автоматически инкрементируется}
For i:=0 to 255 do begin
Pal[i].R:=port[$3C9];
 Pal[i].G:= port[$3C9];
 Pal[i].B:= port[$3C9];
End;
```

Записать свой вариант палитры можно аналогично чтению, только в роли индексного порта используется 3C8h.

Внутренние регистры контроллера ЭЛТ определяют формат данных в видеопамяти для изображения точек, знаков, курсора и т.п.:

- индексный регистр 3D4h;
- регистр данных 3D5h.

Индекс 8: регистр начальной линии изображения. Актуален и в графических, и в текстовых режимах. Позволяет в текстовых режимах сдвигать изображение не на целую строку, а на ее часть (см. рис. 1.39).

Рис. 1.39. Сдвиг букв на часть высоты строки

В регистре используется 7 младших битов, задают номер линии развертки для верхнего края экрана.

```
For i:=0 to 13 do begin
  Port[$3D4]:=8;
  Port[$3D5]:=i;
End;
```

Индекс 9: высота символов текста в линиях (в точках) – 1, используется 5 битов.

Индекс 12h: высота экрана в линиях (в точках) – 1. Позволяет изменять размер видимой части экрана. Для видеоадаптера EGA регистр завершения отображения вертикальной развертки (VDER) содержит 9, а для VGA – 10 бит. Девятый и десятый биты доступны через дополнительный регистр (OVR).

Пример. Нестандартный текстовый видеорежим 80×50×16. На VGA/SVGA по умолчанию используется 25 строк с символами высотой 16 точек, следовательно, высота экрана = 25×16=400 точек. Если задать высоту шрифта 8 точек, то на экране уместится 400/8=50 строк. Используем вышеописанные регистры:

```
Число_строк:=50;
{Задаем новую высоту символов}
Port[$3D4]:=9; Port[$3D5]:=7;
{Задаем новую высоту экрана}
Port[$3D4]:=$12; Port[$3D5]:=Число_строк*8–256–1;
{В доп.регистр заносим старшие биты числа 399}
Port[$3D4]:=7; OVR:=Port[$3D5];
OVR:=OVR or $01;
Port[$3D4]:=7; Port[$3D5]:=OVR;
{Загружаем шрифт 8x8}
r.ax:=$1123; r.bl:=3; Intr($10,r);
{Информируем видеоBIOS о новых параметрах}
mem[$40:$84]:=Число_строк-1;
```

Внутренние регистры синхронизатора управляют временными параметрами работы видеоконтроллера и доступом к отдельным цветовым слоям:

- индексный порт 3C4h.
- порт данных 3C5h.

Индекс 2: регистр разрешения записи цветового слоя. Применяется для рисования в многослойных режимах (например, в 12h). Используются 4 младших бита.

7	6	5	4	3	2	1	0
				X	X	X	X

Каждый бит отвечает за свой цветовой слой: если = 1, то при записи в видеопамять данные попадают в этот слой; если = 0, то нет. Если все 4 бита установлены в 1 и мы записываем по какому-то адресу видеопамяти байт, то этот байт запишется сразу во все слои. Чтобы получать цвета, отличные от 0000 или 1111, надо поочередно разрешать-запрещать разные слои и писать в них разные значения.

Внутренние регистры графического контроллера используются для преобразования данных при обмене с видеопамятью:

- индексный порт: 3CEh;
- порт данных: 3CFh.

Индекс 4: регистр разрешения чтения цветового слоя. Применяется для рисования в многослойных режимах (например, в 12h). Два младших бита используются для задания номера читаемого цветового слоя.

7	6	5	4	3	2	1	0
						X	X

Пример. Требуется в видеорежиме 12h поставить точку красного цвета (в стандартной палитре код = 4) с координатами $x = 9$, $y = 2$. По формулам из п.3.2 получаем: Nбайта = 50h × 2 + 9/8 = 161; Nбита = 7 - (9 mod 8) = 6.

```
{Прочитать и сохранить данные из всех слоев для данного адреса видеопамяти}
port[$3CE]:=4;
port[$3Cf]:=0; {Разрешить для чтения 0-ой слой}
B[0]:=mem[$A000:161];
port[$3CE]:=4;
port[$3Cf]:=1; {Разрешить для чтения 1-ый слой}
B[1]:=mem[$A000:161];
port[$3CE]:=4;
port[$3Cf]:=2; {Разрешить для чтения 2-ой слой}
B[2]:=mem[$A000:161];
port[$3CE]:=4;
port[$3Cf]:=3; {Разрешить для чтения 3-ий слой}
B[3]:=mem[$A000:161];
{Записать в слои для 6-го бита код красного цвета=4}
B[0]:=B[0] and $BF; {x0xxxxxx}
B[1]:=B[1] and $BF; {x0xxxxxx}
B[2]:=B[2] or $40;  {x1xxxxxx}
B[3]:=B[3] and $BF; {x0xxxxxx}
{Загрузить модифицированные данные в видеопамять}
port[$3C4]:=2;
port[$3C5]:=1; {Разрешить для записи 0-ой слой}
mem[$A000:161]:=B[0];
port[$3C4]:=2;
port[$3C5]:=2; {Разрешить для записи 1-ый слой}
mem[$A000:161]:=B[1];
port[$3C4]:=2;
port[$3C5]:=4; {Разрешить для записи 2-ой слой}
mem[$A000:161]:=B[2];
port[$3C4]:=2;
port[$3C5]:=8; {Разрешить для записи 3-ий слой}
mem[$A000:161]:=B[3];
```

Глава 2. АЛГОРИТМЫ ПОСТРОЕНИЯ И ПРЕОБРАЗОВАНИЯ ИЗОБРАЖЕНИЙ

Алгоритмы построения и преобразования изображений базируются в основном на математическом аппарате, изучаемом в рамках дисциплин «Линейная алгебра» и «Аналитическая геометрия».

2.1. Растровые алгоритмы построения геометрических фигур

2.1.1. Отрезки прямой линии

Существует два способа аналитического задания прямой линии:

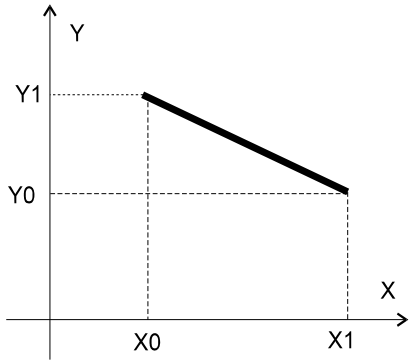


Рис. 2.1. Отрезок прямой

- в виде функции

$y=a\cdot x+b$;

- в виде координат пары точек ((x_0, y_0), (x_1, y_1)) (см. рис. 2.1).

Связь этих двух способов:

$a = (y_1 - y_0)/(x_1 - x_0)$;

$b = y_0 - a\cdot x_0$.

«Естественный» алгоритм рисования прямой использует представление прямой в виде функции

$y = a\cdot x + b$:

```
For x:=x0 to x1 do Точка(x, x*a+b);
```

Этот алгоритм нужно уточнить на ряд случаев:

1) если $|x_1-x_0|<|y_1-y_0|$, то изменять надо координату y, а не x;

2) если $x_1<x_0$ (или $y_1<y_0$), то надо уменьшать координату x (или y), т.е. следует организовать цикл:

```
For x:=x1 to x0 Точка(x, x*a+b);
```

Инкрементный алгоритм Брезенхама (*Brezenham*) выбирает оптимальные растровые координаты для представления отрезка. Основная цель инкрементных алгоритмов – это построение циклов вычисления координат на основе только «быстрых» целочисленных операций (сложения, вычитания, сравнения) без использования «медленных» умножения и деления.

В процессе работы одна из координат – либо x, либо y (в зависимости от углового коэффициента) – изменяется на единицу. Изменение другой координаты (на 0 или 1) зависит от расстояния между действительным положением отрезка и ближайшими координатами сетки. Такое расстояние назовем ошибкой – e. Алгоритм построен так, что требуется проверять лишь знак этой ошибки.

На рис. 2.2 приведена иллюстрация работы алгоритма Брезенхама для отрезка в первом октанте, т.е. когда угловой коэффициент принимает значения от 0 до 1.

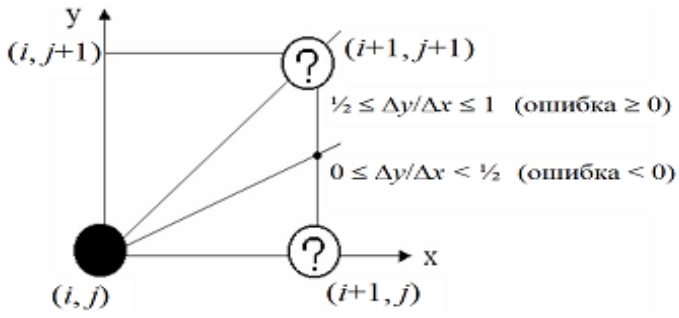


Рис. 2.2. Основная идея алгоритма Брезенхама рисования прямой

Анализируя рисунок, можно заметить, что если угловой коэффициент отрезка ($m=\Delta y/\Delta x$) находится в диапазоне [1/2; 1], то пересечение отрезка с прямой $x=i+1$ будет находиться ближе к узлу $(i+1, j+1)$, чем к узлу $(i+1, j)$. Следовательно, точка растра $(i+1, j+1)$ лучше аппроксимирует ход отрезка, чем точка $(i+1, j)$. Если угловой коэффициент меньше 1/2, то верно обратное, т.е. необходимо выбрать точку растра $(i+1, j)$. Для углового коэффициента, равного 1/2, нет какого-либо предпочтительного выбора, это решение принимает программист. Примем решение выбирать «верхнюю» точку в данном «спорном» случае.

Так как желательно проверять только знак ошибки, то она первоначально устанавливается равной $(m-1/2)$. На каждой итерации величина ошибки поправляется с учетом того, увеличилась координата y на предыдущем шаге или нет. Если ошибка e отрицательна, то отрезок пройдет ниже середины пиксела. Поэтому y не увеличивается. Величина ошибки в следующей точке растра в этом случае вычисляется как

$$e = e + m.$$

Если e положительна, т.е. отрезок пройдет выше середины пиксела. Координата y увеличивается на 1. Ошибка корректируется следующим образом:

$$e=e + m - 1.$$

На рис. 2.3 приведена схема алгоритма Брезенхема для первого октанта. Предполагается, что концы отрезка (x_0, y_0) и (x_1, y_1) не совпадают.

В качестве иллюстрации работы данного алгоритма построим отрезок из точки (0, 0) в точку (5, 4). В данном случае $dx=5$, $dy=4$. Трассировка работы алгоритма приведена в табл. 2.1, а результат работы – на рис. 2.4.

Данный алгоритм нужно уточнить и дополнить для случаев, когда угловой коэффициент $\Delta y/\Delta x$ больше 1 и когда $x_1<x_0$ или $y_1<y_0$.

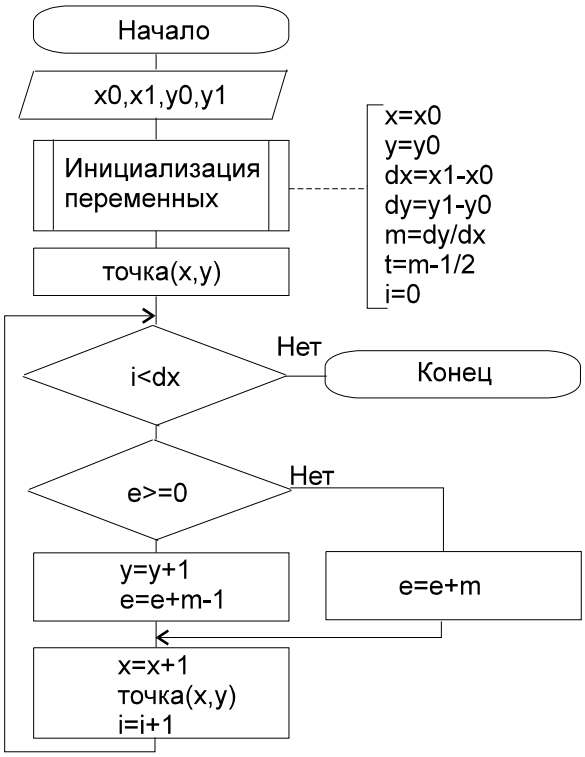


Рис. 2.3. Схема алгоритма Брезенхама рисования прямой

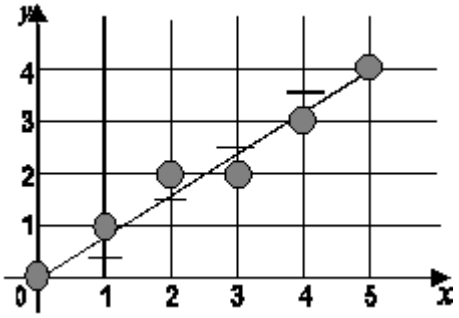


Рис. 2.4. Результат работы алгоритма Брезенхама

Т а б л и ц а 2.1. ***Трассировка работы алгоритма Брезенхама***

i	Точка	e	x	y
0	(0, 0)	3/10	0	0
1	(1, 1)	1/10	1	1
2	(2, 2)	-1/10	2	2
3	(3, 2)	7/10	3	2
4	(4, 3)	5/10	4	3
5	(5, 4)	3/10	5	4

2.1.2. Окружность и эллипс

«Естественный» алгоритм рисования использует представление окружности в виде формулы $Y = \pm\sqrt{R^2 - (X - X_0)^2} + Y_0$:

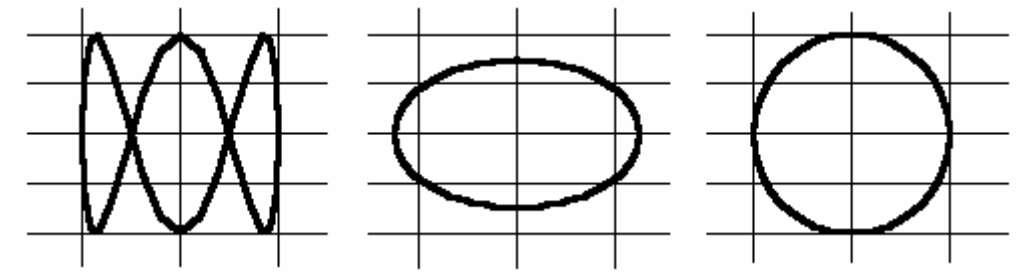
```
For X:=X0-R to X<X0+R do begin
        Точка(X, Y0+sqrt(R*R-sqr(X-X0)));
        Точка(X, Y0-sqrt(R*R- sqr(X-X0)));
End;
```

Параметрический способ рисования использует представление кривой в виде пары функций, зависящих от параметра *t*:

$x=x_0+R_1\cdot\cos(\omega_1\cdot t)$;

$y=y_0+R_2\cdot\sin(\omega_2\cdot t)$;

Здесь $t \in [0..2\pi]$. Если $R_1=R_2$ и $\omega_1=\omega_2$, то получается окружность; если $R_1 \neq R_2$, то получается эллипс; если $\omega_1 \neq \omega_2$, то получаются «фигуры Лиссажу» (см. рис. 2.5).



а) фигура Лиссажу б) эллипс в) окружность

Рис. 2.5. Параметрически заданные кривые

Ниже приведен пример программы на Паскале, реализующей построение окружности по приведенным формулам. В примере используются процедуры moveto и *lineto*, первая из которых устанавливает графический курсор на место с указанными координатами, а вторая – проводит линию от текущего положения курсора до точки, координаты которой являются ее параметрами.

```
procedure my_circle(x,y,r:integer);
 var alfa: integer;
 begin
  moveto(x+r,y);
  for alfa:=1 to 360 do
  { угол измеряется в градусах }
  lineto(round(x+r*cos(alfa*pi/360)),
        round(y+r*sin(alfa*pi/360)))
  { здесь используется радианная мера углов }
 end;
```

Это довольно медленный алгоритм рисования окружности.

Алгоритм Брезенхама построения окружности не требует выполнять деление или вычисление чисел с плавающей точкой, и поэтому работает гораздо быстрее «естественного». Основной идеей алгоритма Брезенхама является регистрация средних значений погрешностей между идеальным положением каждой точки и той позицией на экране дисплея, в которой она действительно отображается.

В алгоритме Брезенхама используются две идеи, позволяющие ускорить построение окружности. Первая: нет необходимости строить всю окружность, достаточно построить некоторую ее часть и последовательным применением преобразований симметрии получить из нее полную окружность. Мы станем строить 1/8 часть окружности, заключенную в сегменте AOB.

Каждая точка этого фрагмента должна быть еще семь раз отображена с помощью преобразований симметрии для получения полной окружности (см. рис. 2.6).

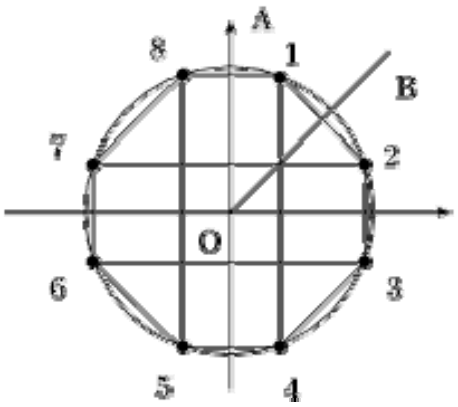


Рис. 2.6. Симметричное отражение точки окружности

Процедура Draw8Pixels выводит 8 симметричных точек окружности, кроме того, именно эта процедура отвечает за расположение центра окружности. Главная процедура вполне может считать, что она строит окружность с центром в начале координат.

```
Procedure Draw8Pixels;
begin
  Точка(x+x0, y+y0, color);
  Точка(x+x0, -y+y0, color);
  Точка(-x+x0, y+y0, color);
  Точка(-x+x0, -y+y0, color);
  Точка(y+x0, x+y0, color);
  Точка(y+x0, -x+y0, color);
  Точка(-y+x0, x+y0, color);
  Точка(-y+x0, -x+y0, color);
end;
```

Приступим к разбору второй – ключевой идеи алгоритма.

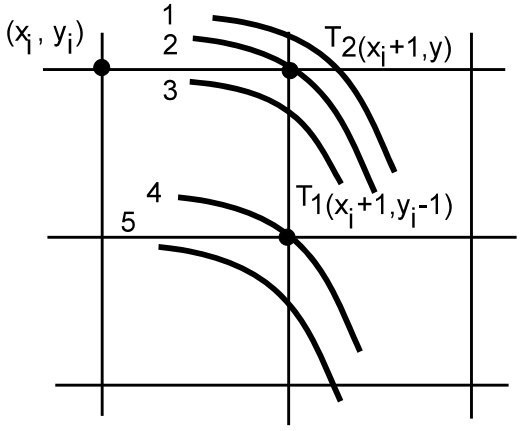


Рис. 2.7. Возможное расположение окружности
относительно пикселов экрана

Пусть мы находимся в некоторой промежуточной фазе построения, только что поставили точку (x_i, y_i) и теперь должны сделать выбор между точками T_1 (x_i+1, y_i-1) и T_2 (x_i+1, y_i). (Так как мы строим часть окружности, заключенную в

сектор AOB, следовательно, подняться выше мы не можем и спуститься вниз более чем на одну точку не можем тоже.)

Реальная окружность может быть расположена относительно точек T_1 и T_2 одним из пяти способов 1-5 (см. рис. 2.7).

Рассчитаем квадраты расстояний от центра окружности до точек T_1 и T_2:

- $R_1^2 = (x_i+1)^2+(y_i-1)^2$,
- $R_2^2 = (x_i+1)^2+(y_i)^2$.

Рассмотрим две погрешности Δ^i_{T1} и Δ^i_{T2}:

- $\Delta^i_{T1} = R_1^2 - R^2 = (x_i+1)^2+(y_i-1)^2 - R^2$,
- $\Delta^i_{T2} = R_2^2 - R^2 = (x_i+1)^2+(y_i)^2 - R^2$

и контрольную величину $\Delta^i = \Delta^i_{T1}+\Delta^i_{T2}$.

При выборе точки, следующей за (x_i, y_i), станем руководствоваться следующим критерием:

- если $\Delta^i > 0$, выберем точку T_1;
- если $\Delta^i \le 0$, выберем точку T_2.

Обоснуем разумность такого выбора. Рассмотрим знаки погрешностей Δ^i_{T1} и Δ^i_{T2} и их влияние на знак контрольной величины Δ^i для всех пяти возможных положений окружности.

Для положения 1: $\Delta^i_{T1} < 0$, $\Delta^i_{T2} < 0 \Rightarrow \Delta^i = \Delta^i_{T1}+\Delta^i_{T2} < 0 \Rightarrow$ выбирается T_2.

Для положения 2: $\Delta^i_{T1} < 0$, $\Delta^i_{T2} = 0 \Rightarrow \Delta^i < 0 \Rightarrow$ выбирается T_2.

Для положения 3 возможны варианты (учитывая, что $\Delta^i_{T1} < 0$, $\Delta^i_{T2} > 0$):

- вариант 3.1: $|\Delta^i_{T1}| \ge |\Delta^i_{T2}| \Rightarrow \Delta^i < 0 \Rightarrow$ выбирается T_2;
- вариант 3.2: $|\Delta^i_{T1}| < |\Delta^i_{T2}| \Rightarrow \Delta^i > 0 \Rightarrow$ выбирается T_1.

Для положения 4: $\Delta^i_{T1} = 0$, $\Delta^i_{T2} > 0 \Rightarrow \Delta^i > 0 \Rightarrow$ выбирается T_1.

Для положения 5: $\Delta^i_{T1} > 0$, $\Delta^i_{T2} > 0 \Rightarrow \Delta^i > 0 \Rightarrow$ выбирается T_1.

Получим выражение для контрольной величины Δ^i:

$\Delta^i = \Delta^i_{T1}+\Delta^i_{T2} = (x_i+1)^2+(y_i-1)^2 - R^2+(x_i+1)^2+(y_i)^2 - R^2 = 2x_i^2+2y_i^2+4x_i-2y_i+3-2R^2$.

Выражение для Δ^{i+1} существенным образом зависит от выбора точки на предыдущем шаге. Необходимо рассмотреть два случая: $y_{i+1} = y_i$ и $y_{i+1} = y_i-1$.

Δ^{i+1} [при $y_{i+1} = y_i$] $= 2x^2_{i+1}+2y^2_{i+1}+4x_{i+1}-2y_{i+1}+3-2R^2 =$

$= 2(x_i+1)^2+2y_i^2+4(x_i+1)-2y_i+3-2R^2 = \Delta^i+4x_i+6.$

Δ^{i+1} [при $y_{i+1} = y_i-1$] $= 2x^2_{i+1}+2y^2_{i+1}+4x_{i+1}-2y_{i+1}+3-2R^2 =$

$= 2(x_i+1)^2+2(y_i-1)^2+4(x_i+1)-2(y_i-1)+3-2R^2 = \Delta^i +4(x_i-y_i)+10.$

Теперь, когда получено рекуррентное выражение для Δ^{i+1} через Δ^i, остается получить Δ^1 (контрольную величину в начальной точке). Величина Δ^1 не может быть получена рекуррентно, ибо не определено предшествующее значение, зато легко может быть найдена непосредственно

$x_1 = 0, y_1 = R \Rightarrow$
$\Delta^1_{T1} = (0+1)^2+(R-1)^2-R^2 = 2-2R,$
$\Delta^1_{T2} = (0+1)^2+R^2-R^2 = 1,$
$\Delta^1 = \Delta^1_{T1} +\Delta^1_{T2} = 3-2R.$

Таким образом, алгоритм построения окружности основан на последовательном выборе точек. В зависимости от знака контрольной величины Δ^i выбирается следующая точка и нужным образом изменяется сама контрольная величина. Процесс начинается в точке $(0, R)$, а первая точка, которую ставит процедура Draw8Pixels, имеет координаты $(x_0, y_0 + R)$. При $x = y$ процесс заканчивается.

Схема алгоритма приведена на рис. 2.8.

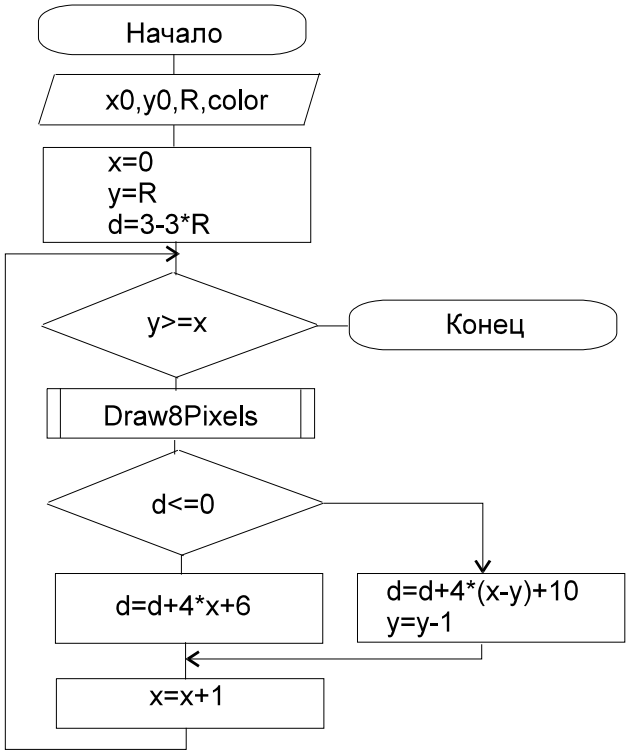


Рис. 2.8. Схема алгоритма Брезенхама рисования окружности

Также существует разновидность алгоритма Брезенхама для рисования эллипсов без использования вещественной арифметики.

2.1.3. Кривые и поверхности Безье

Разработаны математиком Пьером Безье (*Bezier*). Кривые и поверхности Безье были использованы в 60-х годах компанией «Рено» для компьютерного проектирования формы кузовов автомобилей. В настоящее время они широко используются в компьютерной графике.

Кривые Безье описываются *в параметрической форме* следующими формулами:

$$x = P_x(t) = \sum_{i=0}^{m} C_m^i t^i (1-t)^{m-i} x_i ,$$

$$y = P_y(t) = \sum_{i=0}^{m} C_m^i t^i (1-t)^{m-i} y_i ,$$

где $C_m^i = \dfrac{m!}{i!(m-i)!}$ – сочетание m по i, а x_i и y_i – координаты точек-ориентиров. Значение m на единицу меньше количества точек-ориентиров. Рассмотрим примеры кривых Безье при разных значениях m.

1. m=1. (2 точки). Кривая вырождается в отрезок прямой линии (рис.2.9).

$P(t) = (1-t)\cdot P_0 + t\cdot P_1$.

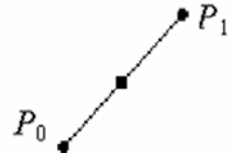


Рис. 2.9. Кривая Безье 1-го порядка

2. m=2. (3 точки) (рис.2.10).

$P(t) = (1-t)^2\cdot P_0 + 2t\,(1-t)\cdot P_1 + t^2\cdot P_2$.

Рис. 2.10. Кривая Безье 2-го порядка

3. m = 3. (4 точки) (рис.2.11.)

$P(t) = (1-t)^3\cdot P_0 + 3t(1-t)^2\cdot P_1 + 3t^2(1-t)\cdot P_2 + t^3\cdot P_3$.

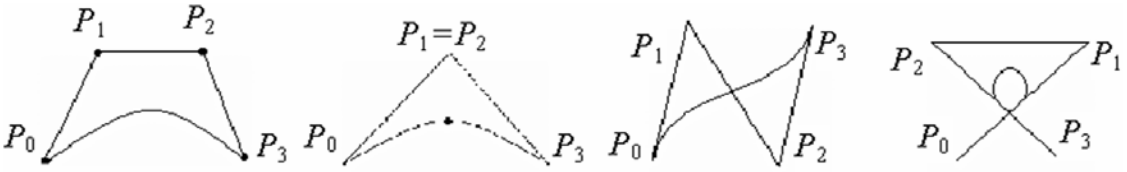


Рис. 2.11. Кривая Безье 3-го порядка

Геометрический алгоритм построения кривой Безье позволяет вычислять координаты x и y по значению параметра t:

1) каждая сторона многоугольника делится пропорционально t (см. рис. 2.12);

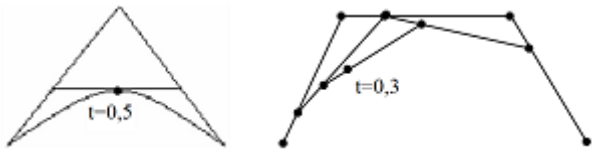


Рис. 2.1.2. «Геометрический» способ построения кривых Безье

2) точки деления соединяются отрезками прямых, образуя новый многоугольник с числом сторон на единицу меньшим;

3.) стороны нового контура снова делятся пропорционально *t* и соединяются до тех пор, пока не будет получена единственная точка деления. Это и будет точка, принадлежащая кривой Безье.

2.2. Растровые алгоритмы закрашивания фигур

Будем считать, что фигура ограничена *контуром*, т.е. непрерывной линией, цвет которой отличен от цвета внутренней части фигуры.

Рис. 2.13. К вопросу о связности

Связность. Будем называть два пиксела связными, если они являются «смежными» (см. рис. 2.13).

4-смежность. Два пиксела являются 4-смежными, если они расположены рядом по горизонтали или вертикали.

8-смежность. Два пиксела являются 8-смежными, если они расположены рядом по горизонтали, вертикали или диагонали.

Существуют универсальные алгоритмы закрашивания «*полигонов*», т.е. многоугольников, заданных произвольным количеством вершин, но у них есть недостатки, связанные с неоднозначностью представления (см. 2.14).

Рис. 2.14. Все эти фигуры заданы одним и тем же множеством вершин

Поэтому стараются сводить задачу закрашивания сложных фигур к закрашиванию множества простых выпуклых фигур (например, треугольников).

2.2.1. Рекурсивный алгоритм с «затравкой»

Шаг 1. Ставится точка нужного цвета («затравка») где-то внутри фигуры, она считается текущей.

Шаг 2. Проверяются поочередно все «соседи» для текущей точки, и если их цвет отличается от цвета закраски и цвета контура, то каждая такая точка закрашивается и сама становится текущей (рекурсивный переход на шаг 1).

Соседние точки определяются в зависимости от того, какое условие связности определено в условие задачи.

Этот алгоритм рекурсивный и приводит к большим затратам стековой памяти, поэтому его целесообразно использовать только для закраски небольших фигур (не более 1 тыс. пикселов).

Модифицированный рекурсивный алгоритм с «затравкой». Заметим, что на каждой строке множество точек, подлежащих закраске, состоит из интервалов, принадлежащих внутренности области. Эти интервалы отделены друг от друга интервалами из точек, принадлежащих границе или внешности области. Кроме того, если набор точек образует связный интервал, принадлежащий

внутренней части области, то точки над и под этим интервалом либо являются граничными, либо принадлежат внутренней части области. Последние могут служить затравочными для строк, лежащих выше и ниже рассматриваемой строки. Суммируя все это, можно предложить следующий алгоритм.

Шаг 1. Ставится точка нужного цвета внутри закрашиваемой области, она считается текущей.

Шаг 2. Заполняем максимально возможный интервал, в котором находится точка, вправо и влево вплоть до достижения граничных точек.

Шаг 3. Запоминаем крайнюю левую x_l и крайнюю правую x_r абсциссы заполненного интервала.

Шаг 4. В соседних строках над и под интервалом (x_l,x_r) находим не заполненные к настоящему моменту внутренние точки области, которые объединены в интервалы, и в крайней правой точке каждого такого интервала рекурсивно переходим к шагу 1.

Алгоритм правильно заполняет любую область, даже такую сложную, как на рис. 2.15.

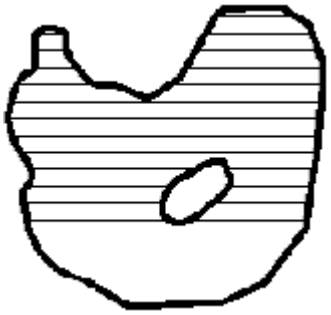


Рис. 2.15. Работа модифицированного рекурсивного алгоритма с «затравкой»

2.2.2. Заполнение полигонов

Контур полигона (многоугольника) определяется вершинами, которые соединены отрезками прямых. Основная идея алгоритма – закрашивание фигур отрезками прямых линий.

Алгоритм заполнения полигонов отрезками прямых следующий:

1) Найти *miny* и *maxy* среди всех вершин P_i.
2) Выполнить цикл по *y* от *min* до *max*.
{
3) Нахождение точек пересечения всех отрезков контура с горизонталью *y*. Координаты x_j точек пересечения записать в массив.
4) Сортировка массива $\{x_j\}$ по возрастанию.
5) Вывод горизонтальных отрезков с координатами:
(x_0,y)-(x_1,y)
(x_2,y)-(x_3,y)
...
(x_{2k},y)-(x_{2k+1},y)
}

В данном алгоритме используется свойство топологии контура фигуры: любая прямая линия пересекает замкнутый контур четное количество раз.

Координата пересечения отрезков p_ip_k с горизонталью y вычисляется по формуле:

$$x=x_i+(y_k-y)(x_k-x_i)/(y_k-y_i).$$

2.2.3. Заполнение областей узорами

Пусть нужный узор хранится в массиве *byte pattern*[*Width*][*Height*].

Пиксельная карта в массиве pattern[][] может быть целым изображением, которым мы хотим раскрасить заданную область, или это может быть маленький *элемент мозаики*, который следует выкладывать с повторением (см. рис. 2.16).

Для обеспечения повторяемости узора пикселу с координатами (x, y) нужно присваивать цвет pattern[x mod Width][y mod Height].

Рис. 2.16. Заполнение области узором

2.3. Аффинные преобразования

Аффинные (т.е. линейные) преобразования служат основным инструментом иллюстративной графики для описания формы объектов, их размеров и размещения на плоскости и в пространстве. Они широко применяются в таких библиотеках, как OpenGL и MS DirectDraw/Direct3D.

2.3.1 Аффинные преобразования на плоскости

Одной из типовых задач, которую приходится решать средствами растровой изобразительной графики, является преобразование как всего изображения целиком, так и его отдельных фрагментов, как то: перемещение, поворот вокруг заданного центра, изменение линейных размеров и т.п. Дж. Форрестом (*Forrest*) разработана система преобразований графических объектов, выполняемых в матричной форме.

Рассмотрим *аффинные* преобразования на плоскости. В общем виде они описываются следующими формулами:

$$\left\{\begin{matrix} X = Ax + By + C \\ Y = Dx + Ey + F \end{matrix}\right\}, \qquad (2.1)$$

где A, B, C, D, E, F – некие константы. Преобразование (2.1) можно записать в матричной форме:

$$\begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} = \begin{pmatrix} A\,B\,C \\ D\,E\,F \\ 0\;0\;1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

Аффинным преобразованиям может подвергаться как сама система координат, так и объекты на плоскости.

Если преобразование производится над системой координат, то значения (x, y) в выражении (2.1) интерпретируются как координаты точки в старой системе координат, а значения (X, Y) – координаты точки в новой системе координат.

Преобразованиям могут подвергаться не только системы координат, но и объекты (фигуры). При преобразованиях объектов на плоскости за (x', y') примем координаты точки до преобразования, (X', Y') – новые координаты точки после преобразования.

В Приложении А приведены основные типы аффинных преобразований систем координат и объектов на плоскости.

Нетрудно доказать справедливость уравнений преобразования поворота объекта на плоскости. Точки P и Q находятся на расстоянии R от начала координат. Координаты точки $P=(R\cos\theta, R\sin\theta)$. Координаты точки $Q=(R\cos(\theta+\varphi), R\sin(\theta+\varphi))$. Подставив в эти выражения два известных тригонометрических соотношения

$$\cos(\theta+\varphi)=\cos(\theta)\cos(\varphi)-\sin(\theta)\sin(\varphi),$$

$$\sin(\theta+\varphi)=\cos(\theta)\sin(\varphi)+\sin(\theta)\cos(\varphi),$$

получим требуемые уравнения преобразования.

В этом случае координата y каждой точки остается неизменной, а каждая координата x перемещается на величину, которая линейно возрастает с ростом y.

2.3.2. Трехмерные аффинные преобразования

Аналогичные преобразования возможны и в трехмерном пространстве. Общий вид трехмерных аффинных преобразований координат:

$$\begin{cases} X = Ax + By + Cz + D, \\ Y = Ex + Fy + Gz + H, \\ Z = Kx + Ly + Mz + N. \end{cases}$$

Матричная форма:

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} A & B & C & D \\ E & F & G & H \\ K & L & M & N \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}.$$

В Приложении А приведены основные типы трехмерных аффинных преобразований систем координат и объектов.

Из формул для вращения объектов можно заметить, что элемент *–sin* появляется в верхней строке матрицы для *x*-вращения и *z*-вращения, но в нижней строке матрицы для *y*-вращения. Имеет ли *y*-вращение какое-то существенное отличие?

Рассмотрим три оси *x*, *y* и *z* в циклическом порядке: x → y → *z* → x → y.. и т.д. Если мы исследуем вращение вокруг какой-нибудь «текущей» (*current*) оси (*x*, *y* или *z*), то можно указать «предыдущую» (*previous*) и «последующую» (*next*) оси. Например, если текущей является ось *x*, то предыдущая – *z* , а следующая – y. Обозначим буквой *P* – координаты точки до преобразования, а *Q* – координаты точки после преобразования. Нетрудно доказать, что при таких обозначениях во всех трех видах поворотов используются одни и те же уравнения:

$$Q_{curr} = P_{curr},$$

$$Q_{next} = \cos\varphi \cdot P_{next} - \sin\varphi \cdot P_{prev},$$

$$Q_{prev} = \sin\varphi \cdot P_{next} + \cos\varphi \cdot P_{prev}.$$

Рассмотрим *основные свойства аффинных преобразований*.

1. Любое аффинное преобразование может быть представлено как совокупность простейших операций – перемещение, масштабирование, поворот. Преобразование сдвига может быть разложено на комбинацию двух поворотов и масштабирования.

2. При аффинном преобразовании сохраняются прямые линии, параллельность прямых, отношения длин отрезков, лежащих на одной прямой, и отношения площадей фигур.

3. Любое преобразование точки относительно начала координат можно представить как обратное преобразование системы координат. Например, поворот точки на угол φ относительно начала координат можно рассматривать как поворот системы координат на угол -φ.

4. Если производится ряд последовательных аффинных преобразований над системой координат или объектом, то результирующая матрица преобразования находится как произведение матриц элементарных преобразований, перемноженных в обратном порядке.

Пример 1. Пусть над объектом последовательно производятся преобразования, описываемые матрицами *A*, *B* и *C*. Результирующая матрица преобразования находится по формуле:

$$Result = C \times B \times A.$$

Пример 2. Найти матрицу преобразования для отражения объекта относительно оси *x*, оси *y*, начала координат (см. рис. 2.17).

1) $\begin{cases} X = x, \\ Y = -y, \end{cases} \quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

2) $\begin{cases} X = -x, \\ Y = y, \end{cases} \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

3) $\begin{cases} X = -x, \\ Y = -y, \end{cases} \quad \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$

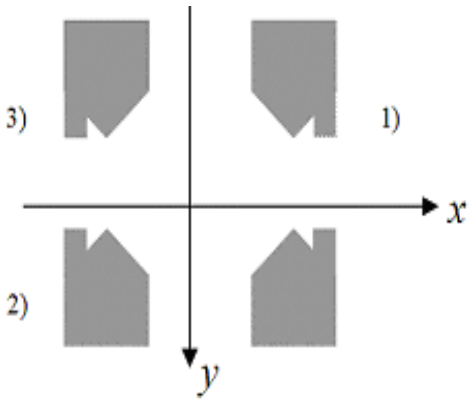


Рис. 2.17. Отражения объекта

Пример 3. Составить преобразование для поворота точки относительно центра с координатами (x_0, y_0) на угол φ.

Это преобразование можно представить в виде комбинации следующих преобразований (см. рис. 2.18):

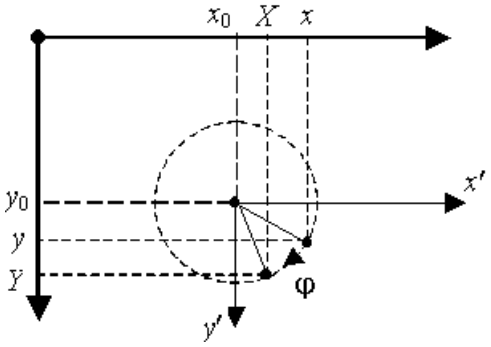


Рис. 2.18. Сложное преобразование

1) перемещение начала системы координат (x, y) в точку (x_0, y_0). Данное преобразование описывается формулами:

$$\begin{cases} x' = x - x_0 \\ y' = y - y_0 \end{cases},$$

2) поворот точки в новой системе координат на угол φ:

$$\begin{cases} X' = x'\cos\varphi - y'\sin\varphi, \\ Y' = x'\sin\varphi + y'\cos\varphi, \end{cases}$$

3) перемещение системы координат на $-x_0$, $-y_0$ назад в (0,0):

$$\begin{cases} X = X' + x_0, \\ Y = Y' + y_0. \end{cases}$$

Итоговая формула:

$$\begin{cases} X = (x - x_0)\cos\varphi - (y - y_0)\sin\varphi + x_0, \\ Y = (x - x_0)\sin\varphi + (y - y_0)\cos\varphi + y_0. \end{cases}$$

Решение задачи в матричной форме:

$$\begin{pmatrix} X \\ Y \\ ! \end{pmatrix} = \begin{pmatrix} \textit{перемещение} \\ \textit{системы координат} \\ \textit{на} - x_0, -y_0 \end{pmatrix} \times \begin{pmatrix} \textit{поворот} \\ \textit{на угол} \\ \varphi \end{pmatrix} \times \begin{pmatrix} \textit{перемещение} \\ \textit{системы координат} \\ \textit{на } x_0, y_0 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} =$$

$$= \begin{pmatrix} 1\,0\,x_0 \\ 0\,1\,y_0 \\ 0\,0\,1 \end{pmatrix} \times \begin{pmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1\,0\,-x_0 \\ 0\,1\,-y_0 \\ 0\,0\quad 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} =$$

$$= \begin{pmatrix} \cos\varphi & -\sin\varphi & -x_0\cos\varphi + y_0\sin\varphi + x_0 \\ \sin\varphi & \cos\varphi & -x_0\sin\varphi + y_0\cos\varphi + x_0 \\ 0 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

2.4. Проецирование объемных фигур на плоскость

Большинство существующих в настоящее время устройств отображения предусматривают построение изображения на плоскости. В математическом смысле *проекции* – это преобразования точек пространства размерности n в точки пространства размерности меньшей, чем n. В компьютерной графике рассматриваются преимущественно проекции образа трехмерного пространства на двумерную картинную плоскость.

Проекция трехмерного объекта, представленного в виде совокупности точек, строится при помощи прямых проецирующих лучей, которые называются проекторами и которые выходят из центра проекции, проходят через каждую точку объекта и, пересекая картинную плоскость, образуют проекцию.

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

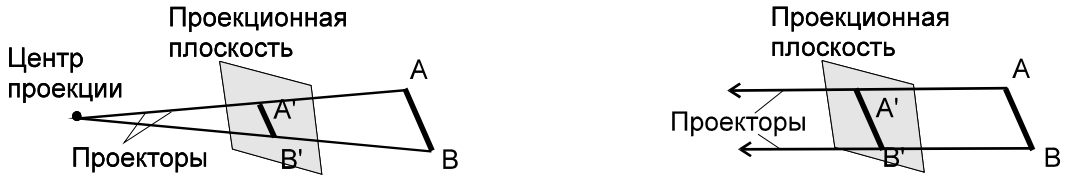
В проецировании участвуют две системы координат:

1) *мировые координаты* – описывают истинное положение объектов в пространстве;

2) *координаты проекции* – описывают изображение объекта в заданной плоскости проецирования.

Основная задача проецирования: для точки с координатами (x, y, z) в пространстве мировых координат найти ее координаты (X, Y, Z) на плоскости проецирования.

Плоские геометрические проекции подразделяются на два основных класса: *центральные* и *параллельные*. Различие между ними определяется соотношением между центром проекции и проекционной плоскостью. Если расстояние между ними конечно, то проекция будет *центральной* (см. рис. 2.19,а), если же оно бесконечно, то проекция будет *параллельной* (см. рис. 2.19,б). Параллельные проекции названы так потому, что центр проекции бесконечно удален и все проекторы параллельны.



а) центральная проекция

б) параллельная проекция

Рис. 2.19. Виды проекций

На рис. 2.20 приведена классификация плоских проекций.

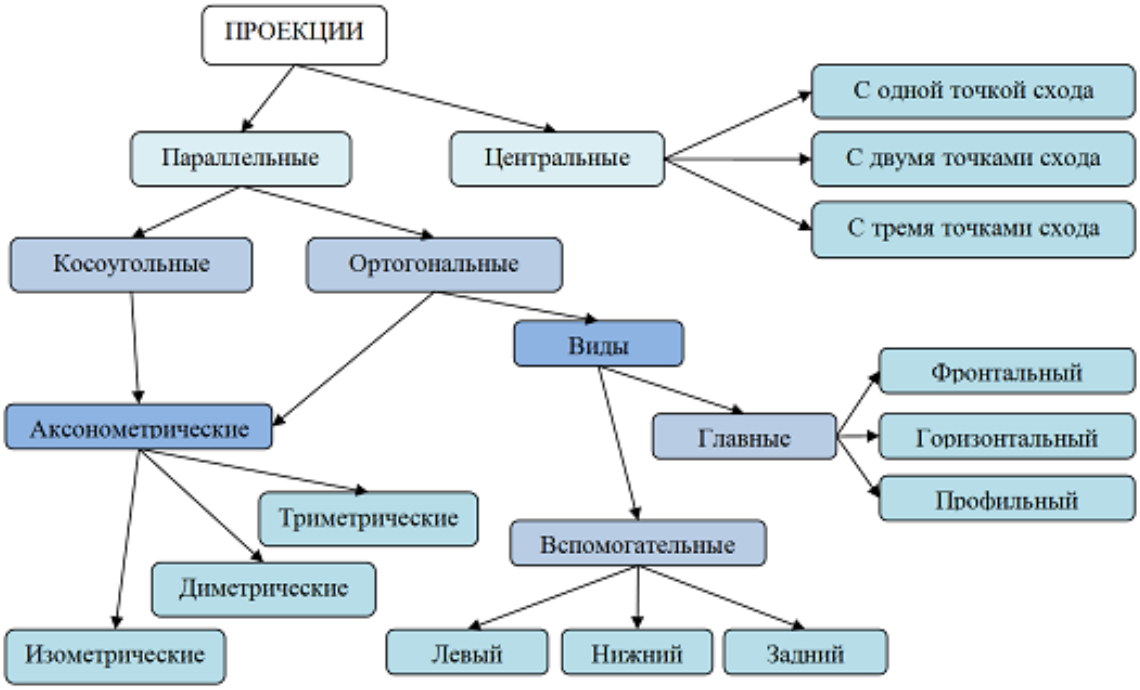


Рис. 2.20. Классификация проекций

Аксонометрическими проекциями называют изображения, полученные путем проецирования параллельными лучами фигуры (предмета) вместе с осями координат на произвольно расположенную плоскость, которую называют аксонометрической (или картинной).

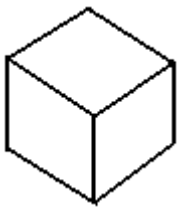
В зависимости от направления проецирующих лучей аксонометрические проекции разделяются на: прямоугольные (проецирующие лучи перпендикулярны плоскости проецирования) и косоугольные (проецирующие лучи наклонены к плоскости проецирования).

В зависимости от наклона осей координат к плоскости проецирования все аксонометрические проекции делятся на три основных вида:

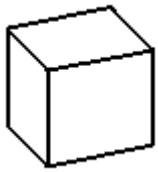
1) *изометрические* (оси *z*, *x* и *y* наклонены одинаково) (см. рис. 2.21,а);

2) *диметрические* (две оси координат имеют один и тот же наклон, а третья – другой) (см. рис. 2.21,б);

3) *триметрические* (все оси имеют разный наклон).



а) изометрическая проекция



б) диметрическая проекция

Рис. 2.21. Виды проекций

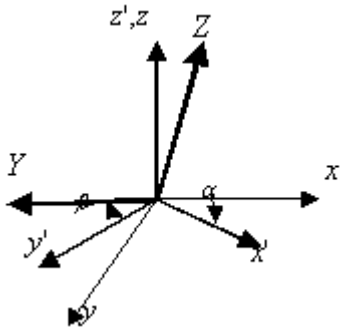


Рис. 2.22. Взаимное положение координат в аксонометрии

При *аксонометрическом проецировании* положение плоскости проецирования в мировых координатах задается двумя углами α и β (см. рис. 2.22):

а) поворот системы координат (x, y, z) относительно оси z мировых координат на угол α. Матрица преобразования данного поворота

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

б) поворот новой системы координат (x', y', z') относительно оси x' на угол β. Данный поворот описывается матрицей

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\beta & \sin\beta & 0 \\ 0 & -\sin\beta & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Итоговое преобразование координат выражается произведением матриц:

$$B \times A = \begin{pmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha\cos\beta & \cos\alpha\cos\beta & \sin\beta & 0 \\ \sin\alpha\sin\beta & -\cos\alpha\sin\beta & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Преобразование в координатной форме:

$$\begin{cases} X = x\cos\alpha + y\sin\alpha, \\ Y = -x\sin\alpha\cos\beta + y\cos\alpha\cos\beta + z\sin\beta, \\ Z = x\sin\alpha\sin\beta - y\cos\alpha\sin\beta + z\cos\beta. \end{cases}$$

Свойства аксонометрической проекции:

1. Аксонометрические проекции параллельных прямых параллельны между собой и в пространстве; очевидно, что если параллельные прямые будут проектирующими, т. е. совпадут с направлением проектирования, то указанное свойство теряет смысл, так как проекциями этих прямых будут точки.

2. Отношение аксонометрических проекций отрезков, расположенных на одной прямой или параллельных прямых, равно отношению самих отрезков.

3. Если линии в пространстве пересекаются (или касаются), то и аксонометрические проекции этих линий пересекаются (или касаются).

4. В общем случае окружность изображается в аксонометрии эллипсом, в частном случае она может проектироваться на плоскость аксонометрических проекций в виде отрезка прямой или окружности.

Перспективное (центральное) проецирование может быть представлено как последовательная комбинация: 1) аксонометрического проецирования; 2) преобразования, корректирующего линейные размеры.

Это преобразование зависит от z_k – мировой координаты z камеры и $z_{\text{пл}}$ – мировой координаты плоскости проецирования (см. рис. 2.23). В координатной форме выглядит следующим образом:

$$\begin{cases} X = x(z_k - z_{пл}) / (z_k - z), \\ Y = y(z_k - z_{пл}) / (z_k - z), \\ Z = z - z_{пл}. \end{cases}$$

Матрица преобразования

$$\begin{pmatrix} \dfrac{(z_k - z_{пл})}{(z_k - z)} & 0 & 0 & 0 \\ 0 & \dfrac{(z_k - z_{пл})}{(z_k - z)} & 0 & 0 \\ 0 & 0 & 1 & -z_{пл} \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Основные свойства перспективного преобразования:

1) не сохраняются отношения длин и площадей;

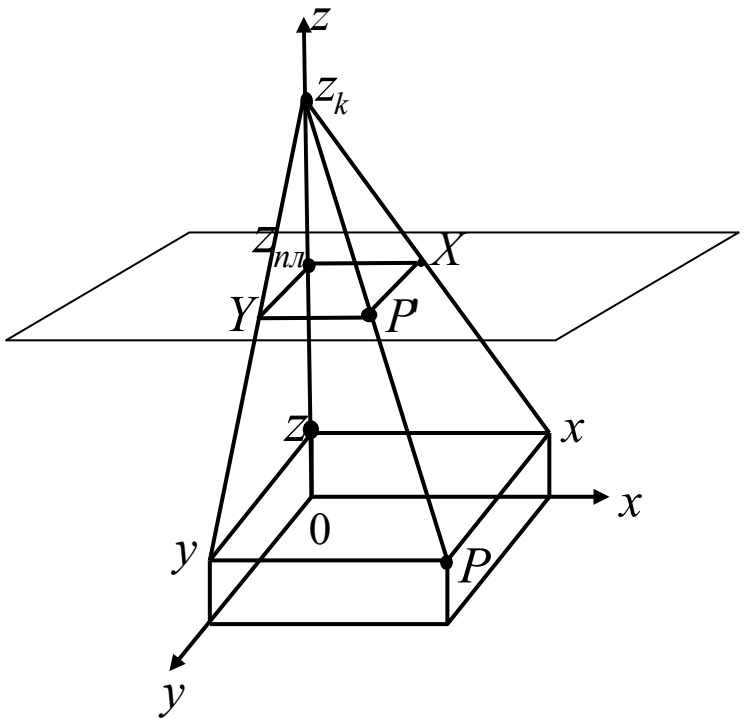


Рис. 2.23. Перспективное проецирование

2) прямые линии изображаются прямыми линиями;

3) параллельные прямые изображаются сходящимися в одной точке.

Отображение в окне. Пусть (X, Y, Z) – координаты проецирования;

($X_э$, $Y_э$, $Z_э$) – экранные координаты.

Для отображения сцены в окне используются два аффинных преобразования – масштабирование и перемещение (см. рис. 2.24).

$$\begin{cases} X_Э = kX + dx, \\ Y_Э = kY + dy, \\ Z_Э = kZ. \end{cases}$$

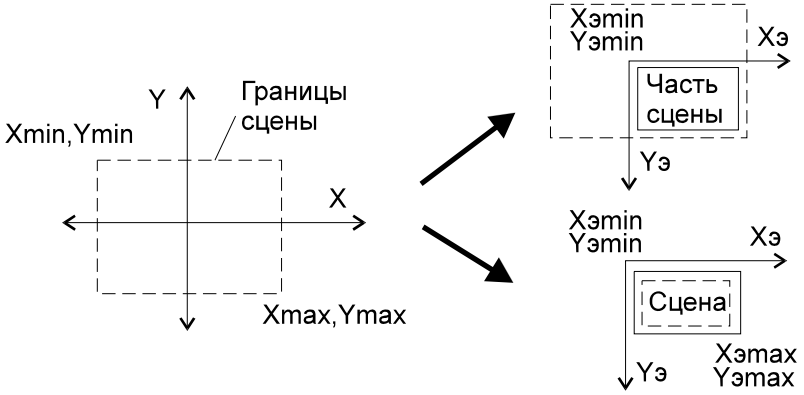


Рис. 2.24. Отображение в окне

Условия вписывания сцены в окно заданных размеров:

$$\begin{cases} X_{Э\min} \le kX_{\min} + dx, \\ Y_{Э\min} \le kY_{\min} + dy, \\ X_{Э\max} \ge kX_{\max} + dx, \\ Y_{Э\max} \ge kY_{\max} + dy. \end{cases}$$

$$\Rightarrow k \le \min\left(\frac{X_{Э\max} - X_{Э\min}}{X_{\max} - X_{\min}}, \frac{Y_{Э\max} - Y_{Э\min}}{Y_{\max} - Y_{\min}} \right) = k_{\min}.$$

Чтобы изображение в окне было максимальным, нужно, чтобы $k = k_{\min}$.

Условия для dx, dy:

$$X_{Э\min} - kX_{\min} \le dx \le X_{Э\max} - kX_{\max},\ Y_{Э\min} - kY_{\min} \le dy \le Y_{Э\max} - kY_{\max}.$$

Чтобы изображение располагалось по центру экрана:

$$dx = \frac{X_{Э\min} - kX_{\min} + X_{Э\max} - kX_{\max}}{2},\ dy = \frac{Y_{Э\min} - kY_{\min} + Y_{Э\max} - kY_{\max}}{2}.$$

2.5. Методы обработки растровых изображений

2.5.1. Преобразование цветного изображения в «серое»

В предположении, что цвет каждого пиксела исходного цветного изображения задан в цветовой схеме RGB, то же самое изображение, исполненное в различных оттенках серого цвета, можно получить в результате преобразования:

$$R'=G'=B'=0{,}3\cdot R + 0{,}59\cdot G + 0{,}11\cdot B,$$

где (R, G, B) – исходные цветовые компоненты пиксела, (R′, G′, B′) – результирующие цветовые компоненты пиксела, соответствующие одной из градаций серого цвета.

Данный алгоритм является характерным представителем так называемых «поэлементных» алгоритмов, то есть тех, которые независимо применяются к каждому элементу (в данном случае – к пикселу) изображения. Многие из рассматриваемых ниже алгоритмов относятся к этому же типу.

2.5.2. Линейное контрастирование

Под «контрастностью» обычно понимается перепад яркостей, присутствующих на изображении. Предположим, что минимально возможное значение яркости пикселов равно $L = 0$, а максимально возможное $H = 63$. Изображение, не использующее всего возможного диапазона яркостей и содержащее яркости пикселов только в пределах от min = 20 до max = 40, очевидно, является «малоконтрастным». Задача контрастирования заключается в «растягивании» реального диапазона яркостей на всю шкалу. Контрастирование можно осуществить при помощи линейного поэлементного преобразования

$$g = af + b. \tag{2.2}$$

Параметры этого преобразования a и b нетрудно определить, исходя из требуемого изменения динамического диапазона. Если в результате обработки нужно получить шкалу $[g_{min}, g_{max}]$, то, как следует из (2.2),

$$g_{\min} = af_{\min} + b,$$

$$g_{\max} = af_{\max} + b.$$

Отсюда

$$a = \frac{g_{\max} - g_{\min}}{f_{\max} - f_{\min}}, \quad b = \frac{g_{\min} f_{\max} - g_{\max} f_{\min}}{f_{\max} - f_{\min}}. \tag{2.3}$$

2.5.3. Пороговая обработка

Пороговая обработка полутонового изображения заключается в разделении всех элементов изображения на два класса по признаку яркости, то есть в выполнении поэлементного преобразования вида:

$$g = \begin{cases} g_{\min}, & f \geq f_0 \\ g_{\max}, & else. \end{cases},$$

где f_0 – некоторое «пороговое» значения яркости. При выполнении пороговой обработки вопрос состоит в выборе порога f_0.

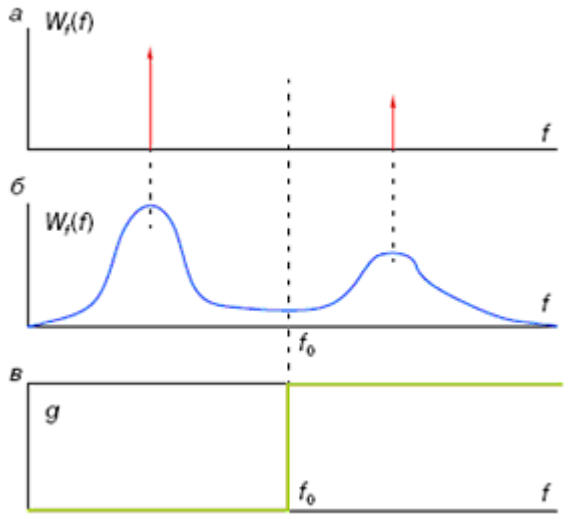


Рис. 2.25. Совместное распределение яркости двух объектов

Пусть полутоновое изображение содержит интересующие нас объекты одной яркости на фоне другой яркости (типичные примеры: машинописный текст или чертежи на фоне бумаги, медицинские пробы под микроскопом на фоне подложки и т.д.). Тогда в идеале плотность распределения яркостей должна выглядеть как две дельта-функции (рис. 2.26,*а*). В данном случае задача установления порога тривиальна: в качестве f_0 можно взять любое значение между «пиками».

На практике, однако, встречаются определенные трудности, связанные с тем, что, во-первых, как для объектов, так и для фона характерен некоторый разброс яркостей. В результате пики функции плотности распределения «расплываются», хотя обычно ее бимодальность сохраняется (рис. 2.26,*б*). В такой ситуации можно выбрать порог f_0, соответствующий положению минимума между модами, то есть использовать функцию поэлементного преобразования, показанную на рис. 2.26,*в*.

2.5.4. Препарирование

Широкий класс процедур обработки изображения заключается в их препарировании, то есть в приведении к такому виду, который, возможно, весьма далек от естественного, но удобен для визуальной интерпретации или дальнейшего машинного анализа.

Частным случаем препарирования является пороговая обработка, рассмотренная выше. Перечислим некоторые другие используемые преобразования.

Очевидным обобщением пороговой обработки является преобразование яркостного среза (рис. 2.26,*а*). Оно позволяет выделить определенный интервал диапазона яркостей входного изображения. Перемещая «рабочий» интервал по шкале и меняя его ширину, можно произвести визуальный анализ отдельных изображенных объектов, различающихся по яркости. Детали, не попадающие в указанный интервал, то есть относящиеся к «фону», будут подавлены.

На рис. 2.26,*б* приведен вариант яркостного среза с сохранением фона. В данном случае изображение в целом сохраняется, но на нем «высвечиваются» участки, попавшие в заданный интервал яркостей.

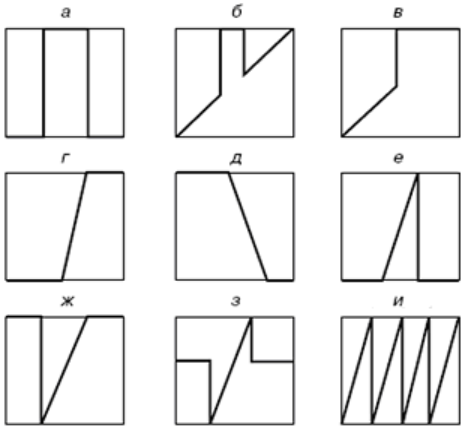


Рис. 2.26. Преобразование яркостного среза

Если этот интервал примыкает к границе шкалы яркости, то получаем преобразование так называемой «неполной» пороговой обработки (рис. 2.26,*в*). Контрастное масштабирование в своем простейшем варианте совпадает по смыслу с линейным контрастированием, рассмотренном в п. 2.5.3, здесь «рабочий» интервал яркостей растягивается на весь диапазон допустимых значений (рис. 2.26, *г*). В других случаях контрастное масштабирование может быть связано с обращением функции яркости, то есть с получением негатива (рис. 2.26,*д*), представлением «рабочего» интервала на однородном фоне: черном (рис. 2.26,*е*), белом (рис. 2.26,*ж*) или сером (рис. 2.26,*з*) и т.д. Пилообразное контрастное масштабирование иллюстрирует рис. 2.26,*и*. Как показывает практика, если изображение состоит из нескольких крупных областей с медленно меняющимися (по плоскости) значениями яркости, то такое преобразование почти не разрушает целостности его восприятия и, в то же время, резко увеличивает контрастность плохо различимых мелких деталей.

2.5.5. Выделение контуров

Задача выделения контуров состоит в построении изображения именно границ объектов и очертаний однородных областей.

На рис. 2.27 *а,б* показаны соответственно исходное изображение, состоящее из областей различной яркости, и его графический вариант, состоящий только из границ этих областей. Будем называть контуром изображения совокупность его пикселов, в окрестности которых наблюдается скачкообразное изменение функции яркости.

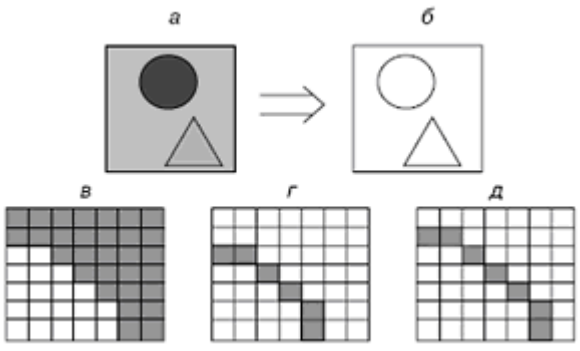


Рис. 2.27. Выделение контуров

Так как цифровой обработке изображение представлено как функция целочисленных аргументов, то контуры представляются линиями шириной, как минимум, в один пиксел. При этом может возникнуть неоднозначность в определении линии контура, как это показано на рис. 2.27,*г,д,* для исходного изображения с перепадом яркости (рис. 2.27,*в*).

Общую процедуру построения бинарного изображения границ объектов иллюстрирует блок-схема, представленная на рис. 2.28.

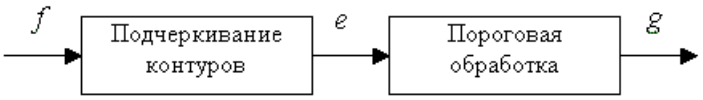


Рис. 2.28. Процедура выделения контуров

Исходное изображение *f* подвергается линейной или нелинейной обработке, с тем чтобы выделить перепады яркости. В результате этой операции формируется изображение *e*, функция яркости которого существенно отличается от нуля только в областях резких изменений яркости изображения *f*. Затем в результате пороговой обработки из изображения *e* формируется графический (контурный) препарат *g*. Правильный выбор порога на втором этапе должен производиться из следующих соображений. При слишком высоком пороге могут появиться разрывы контуров, а слабые перепады яркости не будут обнаружены. При слишком низком пороге из-за шумов и неоднородности областей могут появиться ложные контуры. Других особенностей пороговая обработка не имеет. Поэтому обратим основное внимание на первую операцию – выделение перепадов яркости (контуров) – и рассмотрим основные методы выполнения этой операции.

Градиентный метод. Одним из наиболее простых способов выделения границ является пространственное дифференцирование функции яркости. Градиент функции яркости вычисляется по формуле

$$|\Delta f(x,y)| = \sqrt{\left(\frac{\partial f(x,y)}{\partial x}\right)^2 + \left(\frac{\partial f(x,y)}{\partial y}\right)^2}$$

и пропорционален максимальной (по направлению) скорости изменения функции яркости в данной точке и не зависит от направления контура.

Частные производные можно вычислить приближенно с помощью следующих конечных разностей:

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x_{i+1},y_j) - f(x_i,y_j)}{x_{i+1} - x_i}, \quad \frac{\partial f(x,y)}{\partial y} \approx \frac{f(x_i,y_{j+1}) - f(x_i,y_j)}{y_{i+1} - y_i}.$$

Т. к. $x_{i+1} - x_i = y_{i+1} - y_i = 1$,

то $$\frac{\partial f(x,y)}{\partial x} \approx f(x_{i+1},y_j) - f(x_i,y_j), \quad \frac{\partial f(x,y)}{\partial y} \approx f(x_i,y_{j+1}) - f(x_i,y_j).$$

Следовательно,

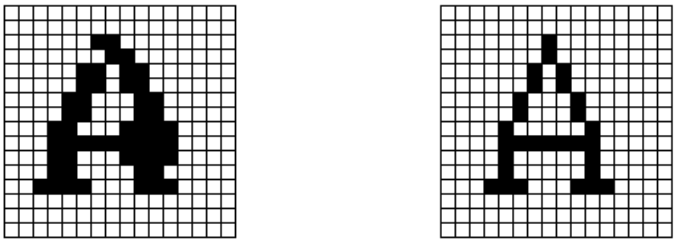
$$|\Delta f(x,y)| \approx \sqrt{\left(f(x_{i+1},y_j)-f(x_i,y_j)\right)^2+\left(f(x_i,y_{j+1})-f(x_i,y_j)\right)^2}\ .$$

Таким образом, операция выделения контуров заключается в выполнении нелинейной локальной обработки изображения «окно» 2×2 (без одной точки):

$$e(i,j)=\sqrt{\left(f(x_{i+1},y_j)-f(x_i,y_j)\right)^2+\left(f(x_i,y_{j+1})-f(x_i,y_j)\right)^2}\ .$$

2.5.6. Алгоритмы утончения линий

Задача «*утончения линий*» (или «*задача скелетизации*») актуальна в тех случаях, когда необходимо преобразовать произвольное монохромное изображение в штриховое, т.е. состоящее из отдельных точек и линий. Например, такая проблема встречается в процессе предварительной обработки отсканированных изображений текстов и чертежей, предназначенных для дальнейшего распознавания (см. рис. 2.29).



а) до утончения б) после утончения

Рис. 2.29. Утончение линий

Существует множество разнообразных алгоритмов, мы рассмотрим наиболее характерные из них.

Итерационные алгоритмы предусматривают несколько последовательных циклов просмотра изображения, во время каждого из которых удаляются пикселы наружного слоя фигур. Различают *последовательные* алгоритмы, в которых важен порядок обхода пикселов изображения (например, слева направо и сверху вниз) и *параллельные*, в которых пикселы могут просматриваться в произвольном порядке. Во всех подобных алгоритмах пикселы сначала только помечаются как удаленные, а убираются с изображения только по окончании цикла сканирования.

В качестве примера рассмотрим параллельный а*лгоритм Зонга–Суэна.*

Введем матрицу 3×3: $\begin{bmatrix} P9 & P2 & P3 \\ P8 & P1 & P4 \\ P7 & P6 & P5 \end{bmatrix}$.

Накладываем матрицу на изображение, совмещая интересующий нас пиксел с $P1$. Каждая итерация состоит из 2 подытераций.

Подытерация 1: пиксел P1 удаляется из изображения, если выполняются следующие условия:

a) $2 \le B(P1) \le 6$;

b) $A(P1) = 1$;

c) $P2 \times P4 \times P6 = 0$;

d) $P4 \times P6 \times P8 = 0$.

Где A(P1) – число конфигураций 01 в последовательности P2, P3, P4, P5, P6, P7, P8, P9, замыкая эту цепочку на P2, т.е. вокруг этого пиксела существует только один переход от 0 к 1.

$$B(P_i) = \sum_{i=2}^{9} P_i .$$

Подытерация 2: выполняется аналогично, только

$$P2 \times P4 \times P8 = 0,$$

$$P2 \times P6 \times P8 = 0.$$

Таким образом, суть подытераций заключается в следующем.

Подытерация 1. Удаление точек на юго-восточной границе и северо-западных угловых точек.

Подытерация 2. Удаление точек на северо-западной границе и юго-восточных угловых точек.

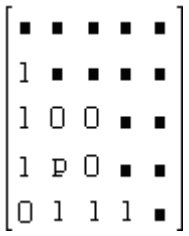


Рис. 2.30. Пикселы изображения

Эти итерации мы выполняем до тех пор, пока возможно удалять пикселы. Но этими условиями мы не охватываем некоторых случаев. Рассмотрим следующую картинку (см. рис. 2.30). Пиксел P, если он был единицей, этими условиями не удаляется. Поэтому проводится еще одна итерация, которая устраняет подобные недочеты.

На этой итерации ищутся два единичных пиксела по вертикали или горизонтали, которые окружены нулями.

Итак, точка P удаляется, если выполняется одно из условий:

1) !P9×P4×P6=1,

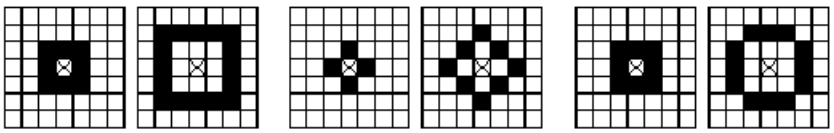
2) !P5×P8×P2=1,

3) !P3×P6×P8=1,

4) !P7×P2×P4=1,

(где !P9 – отрицание P9).

Другой пример – *волновой алгоритм,* использующий последовательную пометку пикселов, причем этот процесс напоминает распространение кругов на воде. На рис. 2.31 изображены первые генерации трех видов «волны», родившейся из центрального пиксела. Под «связностью» здесь понимается количество рассматриваемых «соседей» пиксела.



а) 8-связная б) 4-связная в) комбинированная

Рис. 2.31. Разновидности «волны»

Выпущенная из одной точки и распространяющаяся «волна» обладает рядом замечательных свойств (см. рис. 2.32): довольно быстро приобретает вид «параллельных гребней», умеет «поворачивать» и «огибать препятствия». Идея метода утончения линий состоит в том, чтобы отслеживать середины отрезков, образующих «гребни волн» – именно через них можно провести линию, образующую «скелет» фигуры.

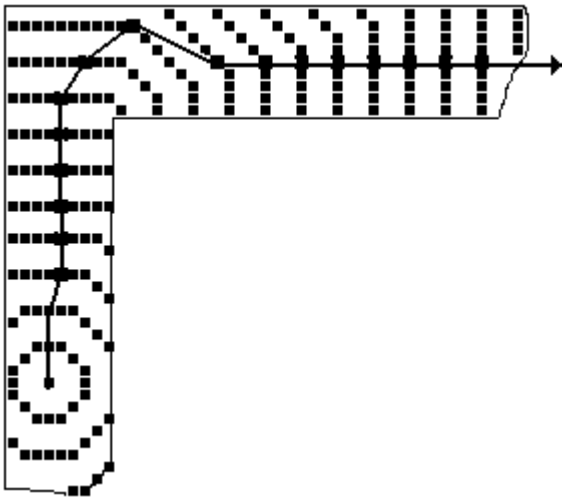


Рис. 2.32. Свойства «волны»

Эвристический алгоритм (от греч. *heurisco* – открывать, узнавать новое) предполагает, что для каждого пиксела $C(x,y)$ с координатами x и y вводятся локальная плотность изображения

$$d(x,y)=\sum_{i=-1}^{1}\sum_{j=-1}^{1}C(x+i,y+j)-C(x,y)$$

и плотность локальных плотностей

$$D(x,y)=\sum_{i=-1}^{1}\sum_{j=-1}^{1}d(x+i,y+j)-d(x,y).$$

Пиксел остается на изображении, если для него $d>F1$ и $D>F2$, где $F1$ и $F2$ – некоторые константы, подбираемые для конкретного изображения экспериментально.

2.5.7. Преобразования изображений при помощи масочных фильтров

Эти методы преобразования изображений, в отличие от поэлементных, предусматривают преобразование значения яркости пиксела, зависящее не только от него самого, но и от его «окружения». В простейшем случае в окружение включаются 8 ближайших пикселов, но может рассматриваться и более обширное «соседство».

Методы *линейной масочной фильтрации* предусматривают, что над яркостью каждого пиксела $C(X,Y)$ производится преобразование

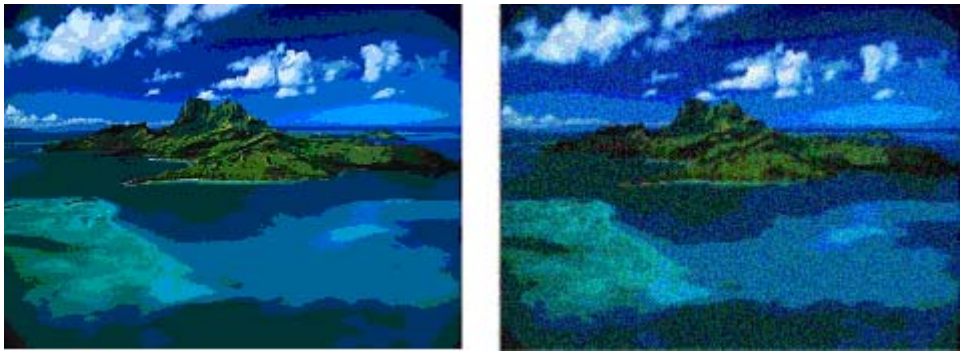
$$C'(X,Y) = A + B \cdot \sum_{i=-N}^{N} \sum_{j=-N}^{N} C(X+i, Y+j) \cdot M(X+i, Y+j),$$

где N – количество «слоев», входящих в окружение пиксела (обычно, N = 1), A и B – некие константы, M – некая матрица («маска») размерности $(2 \cdot N+1) \times (2 \cdot N+1)$. Следует иметь в виду, что при попиксельном сканировании изображения в рассмотрение должны включаться исходные, а не преобразованные значения яркостей пикселов.

1. Сглаживающие маски (см. рис. 2.33):

$$A_1 = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \quad A_2 = \frac{1}{10}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \quad A_3 = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}. \tag{2.4}$$

Коэффициенты масок нормированы, т.е. $\sum_{(k_1,k_2) \in W} \sum a(k_1,k_2) = 1$, с тем чтобы процедура подавления помех не вызывала смещения яркости исходного изображения.



а) исходное изображение б) размытое изображение

Рис. 2.33. Сглаживающие маски

Маски (2.4) отличаются степенью сглаживания шумов (у маски A_1 она максимальная, у A_3 – минимальная). Выбор коэффициентов маски должен производиться экспериментально. При увеличении степени сглаживания шумов происходит также подавление высокочастотной составляющей полезного изображения, что вызывает исчезновение мелких деталей и размазывание контуров. Если требуемая степень сглаживания с применением маски размера 3×3 не достигает, то следует использовать сглаживающие маски больших размеров (5×5,7×7,…).

2. Увеличение контрастности:

$$A = 0,\ B = 1/4,\ M = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 8 & -1 \\ 0 & -1 & 0 \end{bmatrix};\ A = 0,\ B = 1,\ M = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}.$$

3. Придание изображению рельефности (см. рис. 2.34):

$$A = 128\ (32),\ B = 1/2,\ M = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

а) исходное изображение б) рельефное изображение

Рис. 2.34. Придание изображению рельефности

Значение коэффициента A выбирается равным половине от максимального значения яркости. В зависимости от видеорежима может принимать значения 128 или 32.

4. В качестве примера *нелинейной масочной фильтрации* приведем метод *медианной фильтрации*: все пикселы, входящие в окружение текущего (включая и его самого), сортируются по яркости, затем текущий пиксел заменяется центральным пикселом в отсортированной группе. Медианная фильтрация уменьшает резкость и позволяет избавиться от мелких деталей, например, от точечных помех.

2.5.8. Векторизация растровых изображений. Преобразование Хафа

Элементы растеризации векторных изображений были рассмотрены нами выше, в разделе, посвященном построению простых геометрических фигур средствами растровой графики. Гораздо более сложна обратная операция – преобразование растровых изображений в векторные.

Рассмотрим специализированное преобразование Хафа (*Hough*). Согласно методике, использующей это преобразование, на монохромном растровом изображении вместо декартовой системы координат вводится полярная (см. рис. 2.35) с координатами (ρ, θ).

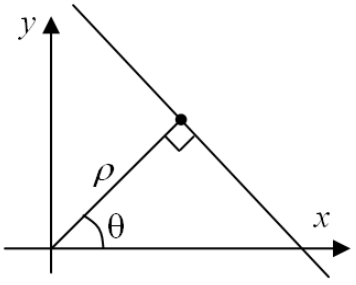
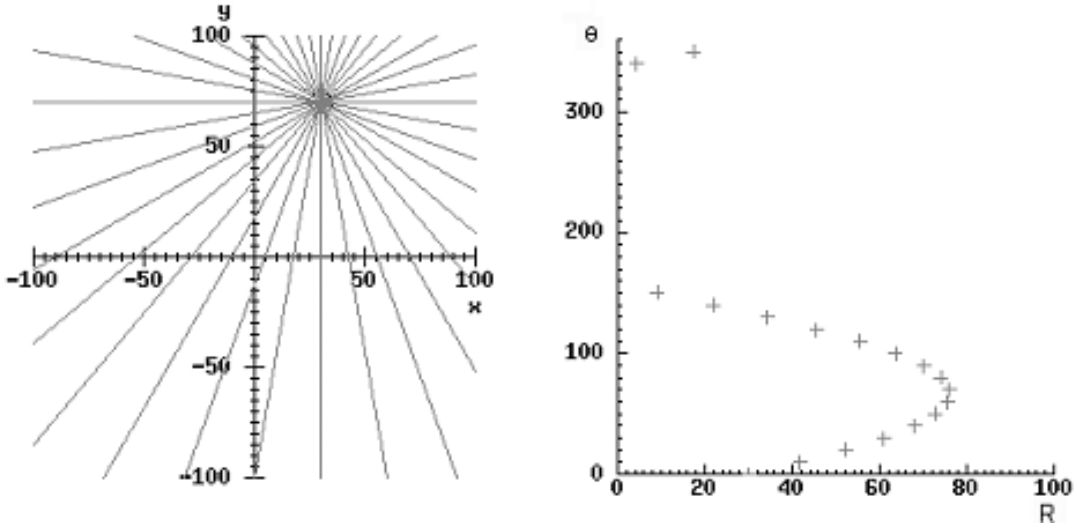


Рис. 2.35. Переход к полярной системе координат

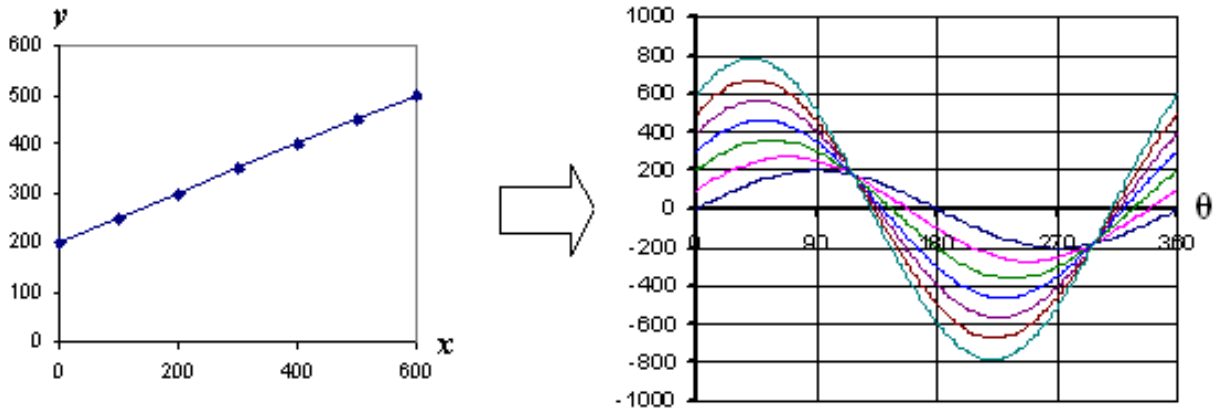
Прямая, проходящая через точку с координатами (x, y), будет определяться

уравнением $x \cdot \cos\theta + y \cdot \sin\theta = \rho$. Функция, задающая семейство прямых, проходящих через эту точку, будет иметь вид $F(\rho, \theta, x, y) = x \cdot \cos\theta + y \cdot \sin\theta - \rho = 0$. Через каждую точку (x, y) растрового изображения можно провести несколько прямых с разными ρ и θ (см. рис. 2.36,*а*). Таким образом, каждой точке (x, y) соответствует некий набор точек в фазовом пространстве (ρ, θ), образующий синусоиду (см. рис. 2.36,*б*). В свою очередь, каждой точке пространства (ρ,θ) соответствует набор точек (x, y) на изображении, образующий прямую. Каждой точке (ρ_0, θ_0) пространства (ρ, θ) можно поставить в соответствие счетчик, соответствующий количеству точек (x, y), лежащих на прямой $x \cdot \cos\theta_0 + y \cdot \sin\theta_0 = \rho_0$.



а) в Декартовой б) в полярной

Рис. 2.36. Семейство прямых в декартовой и полярной системах координат



а) в Декартовой б) набор синусоид

Рис. 2.37. Единичная прямая в разных системах координат

Вследствие дискретности растрового представления исходного изображения каждой прямой в координатах (x, y) будет соответствовать не одна точка, а сгущение точек в координатах (ρ,θ).

Таким образом, достаточно выбрать на изображении, построенном в полярных координатах, самые «жирные пятна» и для их центров произвести преобразование в полярные координаты, получив тем самым параметры соответствующей прямой. Алгоритм преобразования Хафа приведен на рис. 2.38.

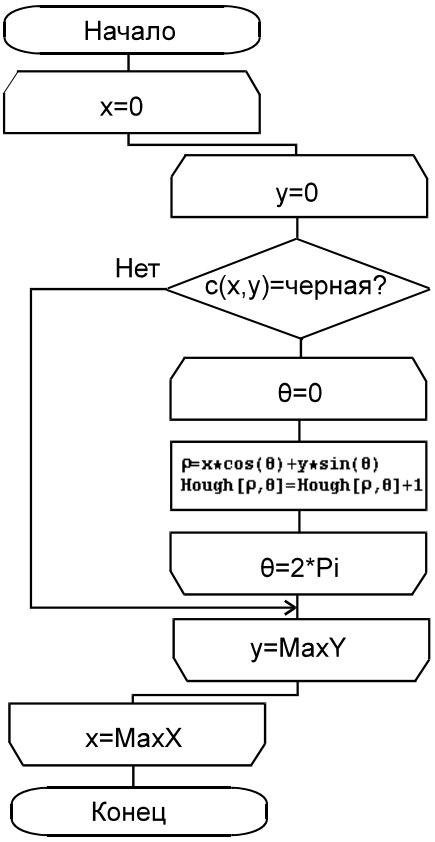


Рис. 2.38. Алгоритм Хафа

Существуют также разновидности преобразования Хафа для дуг и окружностей. Таким образом, преобразование Хафа позволяет представить монохромное штриховое изображение в виде набора формул, описывающих прямолинейные и криволинейные элементы изображения, то есть произвести векторизацию этого изображения.

Глава 3. МЕТОДЫ И АЛГОРИТМЫ ТРЕХМЕРНОЙ ГРАФИКИ

Основная особенность «трехмерной» изобразительной графики заключается в том, что изображения являются проекциями объемных фигур на плоскость. Соответственно, приходится иметь дело с двумя независимыми системами координат: положение объектов рассматривается относительно «мировых» координат, а положение их изображений – относительно «экранных». Основная задача трехмерной графики: зная положение точки в мировой системе координат (X, Y, Z), рассчитать ее экранные координаты (X', Y', Z').

Инструментом решения основной задачи трехмерной графики является аппарат преобразований объектов в трехмерном пространстве.

3.1. Модели описания поверхностей

Для описания формы поверхностей объектов могут использоваться разнообразные методы. Рассмотрим некоторые из них.

3.1.1. Аналитическая модель

Аналитическая модель – это описание поверхности математическими формулами. Например, в виде функции двух аргументов $z = f(x, y)$. Или используя уравнение $F(x, y, z)=0$. Также возможно использование параметрической формы описания поверхности:

$$\begin{cases} x = F_x(s,t), \\ y = F_y(s,t), \\ z = F_z(s,t), \end{cases}$$

где s, t – параметры, которые изменяются в некотором определенном диапазоне.

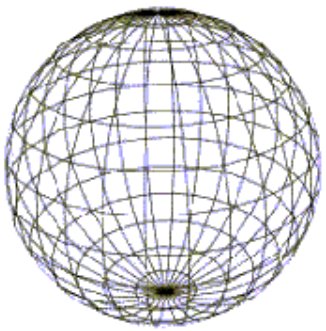


Рис. 3.1. Представление шара

Пример. Поверхность шара можно описать следующим образом (см. рис. 3.1):

1) $z = \pm\sqrt{R^2 - x^2 - y^2}$ – функцией двух аргументов;

2) $x^2 + y^2 + z^2 - R^2 = 0$ – уравнением;

3) параметрически $\begin{cases} x = R\sin\alpha\cos\beta, \\ y = R\sin\alpha\sin\beta, \\ z = R\cos\alpha, \end{cases}$

где $-90^\circ < \alpha < 90^\circ$, $0 < \beta < 360^\circ$.

Для описания сложных поверхностей часто используются сплайны. Сплайн – это специальная функция, более всего пригодная для аппроксимации

отдельных фрагментов поверхности. Чаще всего используют кубические сплайны.

Достоинства аналитической модели:

1) легкость расчета координат каждой точки поверхности, нормали к ней;

2) небольшой объем информации для описания сложных форм.

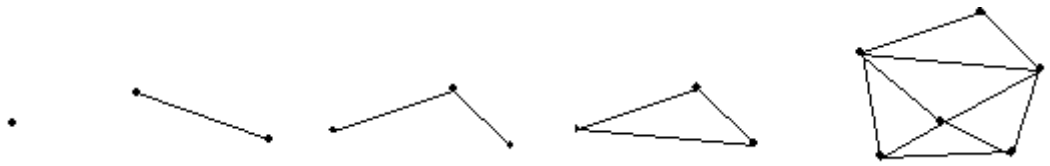
Недостатки аналитической модели:

1) применение сложных формул, что замедляет расчеты;

2) невозможность использования данной формы описания непосредственно для построения изображения поверхности.

3.1.2. Векторная полигональная модель

Термин «векторная», используемый в данном разделе, не имеет никакого отношения к векторным изображением. Просто для описания пространственных объектов в данной модели используются следующие элементы: вершины, полилинии, полигоны, полигональные (многоугольные) поверхности и, наконец, векторы, т.е. отрезки прямых (см. рис. 3.2).



а) вершина б) вектор в) полилиния г) полигон д) поверхность

Рис. 3.2. Элементы описания векторной полигональной модели

Вершина – главный элемент описания. Каждый объект однозначно определяется координатами собственных вершин.

Вершина может моделировать отдельный точечный объект, размер которого не имеет значения, а также может использоваться в качестве конечных точек для линейных объектов и полигонов. Двумя вершинами задается вектор. Несколько векторов задают *полилинию*. Полилиния может представлять собой контур полигона. *Полигон* моделирует площадный объект. Один полигон может описывать плоскую *грань* объемного объекта. Несколько граней составляют объемный объект в виде *полигональной поверхности* – многогранник или незамкнутую поверхность («полигональная сетка»).

Достоинства векторной полигональной модели:

1) удобство масштабирования;

2) небольшой объем данных для описания простых поверхностей;

3) необходимость вычислять только координаты вершин при преобразованиях системы координат или перемещении объекта;

4) аппаратная поддержка многих операций, что обуславливает достаточную скорость для анимации.

Недостатки:

1) сложные алгоритмы визуализации для создания реалистичных изображений;

2) сложные алгоритмы выполнения топологических операций (разрезы);

3) погрешности моделирования при аппроксимации плоскими гранями.

Это метод является наиболее популярным при построении трехмерных изображений.

3.1.3. Воксельная модель

Воксельная модель – это трехмерный растр (см. рис. 3.3). *Воксел* – это элемент объема (от англ. *voxel – volume element*).

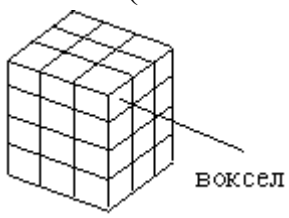


Рис. 3.3. Трехмерный растр

Каждый воксел имеет свой цвет и прозрачность. Полная прозрачность воксела означает пустоту соответствующей точки объема. Чем больше вокселов в определенном объеме и меньше размер вокселов, тем точнее моделируются трехмерные объекты (см. рис. 3.4).

Рис. 3.4. Воксельная модель

Достоинства воксельной модели:

1) позволяет достаточно просто описывать сложные объекты и сцены, простая процедура отображения;

2) простое выполнение топологических операций (например, для выполнения разреза достаточно лишь сделать прозрачными соответствующие вокселы).

Недостатки:

1) большой объем информации, например, объем 256×256×256 имеет небольшую разрешающую способность, но требует более 16 млн. вокселов;

2) значительные затраты памяти ограничивают разрешающую способность, точность моделирования;

3) малая скорость отображения объемных сцен;

4) проблемы при увеличении/уменьшении изображения.

3.1.4. Равномерная сетка

Эта модель заключается в том, что каждому узлу сетки с индексами (i, j) приписывается значение высоты z_{ij}. Расстояние между узлами одинаковое dx, dy (см. рис. 3.5).

Достоинства:

1) простота описания поверхности;

2) возможность быстро узнать высоту любой точки поверхности простой интерполяцией.

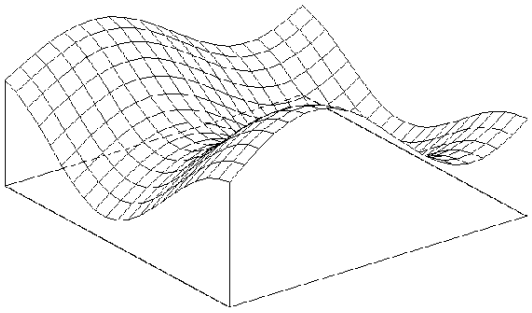


Рис. 3.5. Равномерная сетка

Недостатки:

1) невозможность моделирования поверхностей с неоднозначной функцией высоты;

2) для описания сложных поверхностей необходимо большое количество узлов;

3) описание отдельных типов поверхностей может быть сложнее, чем в других моделях. Например, многогранная поверхность требует избыточного объема данных для описания по сравнению с полигональной моделью.

3.1.5. Неравномерная сетка. Изолинии

Неравномерная сетка – это модель описания поверхности в виде множества отдельных точек $\{(x_0, y_0, z_0), (x_1, y_1, z_1), \ldots, (x_n, y_n, z_n)\}$, принадлежащих поверхности.

Данные точки могут быть логически никак не связаны между собой. А могут представлять собой изолинии – линии одной высоты.

Описание высоты изолиниями используется в картографии (см. рис. 3.6).

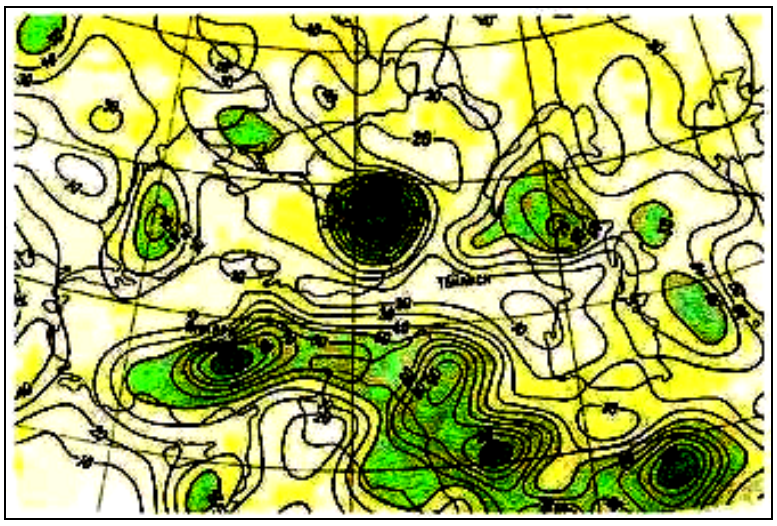


Рис. 3.6. Изолинии

Достоинства:

1) используются отдельные опорные точки, наиболее важные для заданной формы поверхности, что обуславливает меньший объем информации по сравнению с другими моделями (например, с равномерной сеткой);

2) наглядность показа рельефа поверхности изолиниями на картах.

Недостатки:

1) невозможность или сложность выполнения многих операций над поверхностями (интерполяция);

2) использование сложных алгоритмов преобразования в другие формы описания поверхности.

3.2. Визуализация объемных изображений

Разделим способы визуализации по характеру изображений и сложности алгоритма.

1) *Каркасная модель*. Каркас состоит из отрезков прямых или кривых линий, видны все ребра (см. рис. 3.7, *а*).

Для построения такой модели объекта требуется знать координаты всех вершин в мировых координатах.

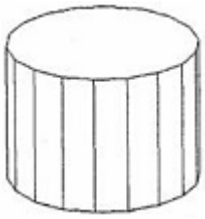
Алгоритм:

- определить экранные координаты для всех вершин в соответствии с выбранным видом проекции;
- вывод всех ребер.

2) *Показ с удалением невидимых линий.* Показ поверхности в виде многогранников с плоскими гранями или сплайнами с удалением невидимых точек (см. рис. 3.7, *б*).

3) *Сложное закрашивание объектов* с имитацией отражения света, затененности, прозрачности, наложение текстур (см. рис. 3.7, *в*).

а) каркасная модель



б) показ с удалением невидимых линий

в) сложное закрашивание

Рис. 3.7. Способы визуализации объемных изображений

3.3. Методы удаления невидимых поверхностей

Рассмотрим несколько типичных методов, разработанных для решения задачи удаления невидимых линий и поверхностей.

1. *«Алгоритм художника»* заключается в том, что объекты трехмерной сцены предварительно сортируются по степени удаленности от «глаза». При

построении изображения в первую очередь «рисуются» удаленные объекты, потом все более и более близкие, благодаря чему «видимые» загораживают «невидимых».

Довольно распространенная характеристика удаленности для грани ABC – это среднее значение z, mid_z = (A.z+B.z+C.z)/3.

При помощи «алгоритма художника» можно строить, например, закрашенные выпуклые объемные фигуры. Цикл изменения координат необходимо запрограммировать так, чтобы сначала изображались «задние», невидимые грани, а лишь потом те, которые видны глазу.

Достоинство «алгоритма художника» – простота. Недостатки: при некотором расположении граней этот алгоритм вообще не может дать правильного результата (см. рис. 3.8 – в каком порядке грани ни рисуй, получится неправильно). При некотором расположении граней и использовании среднего значения z как характеристики удаленности алгоритм тоже дает неправильный результат. Пример приведен на рис. 3.9. В этом случае горизонтальную грань надо отрисовывать второй, но по среднему значению z она лежит дальше, и таким образом получается, что ее надо отрисовывать первой. Возможные пути решения этой проблемы – какие-то изменения характеристики удаленности либо моделирование, не вызывающее таких ситуаций. И, наконец, при использовании этого алгоритма отрисовываются вообще все грани сцены, и при большом количестве загораживающих друг друга граней большая часть времени тратится на отрисовку невидимых в конечном итоге частей. То есть совершенно впустую.

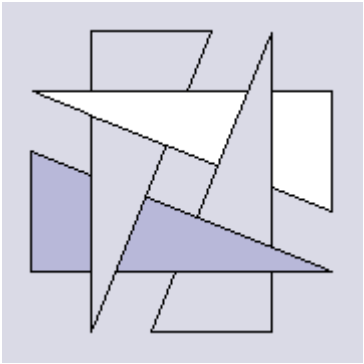


Рис. 3.8. Невозможно изобразить при помощи «алгоритма художника»

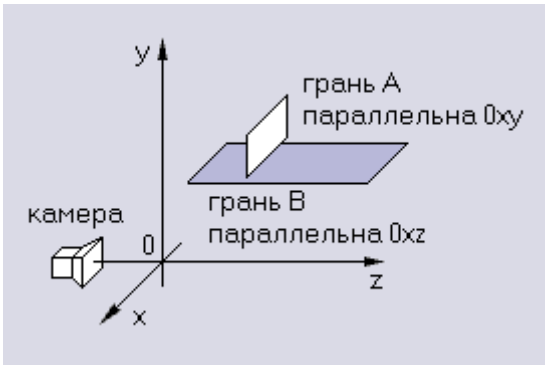


Рис. 3.9. Использование среднего значения z дает неправильный результат

2. *Метод предварительной сортировки граней* разработан специально для полигональных моделей выпуклых фигур. Предварительно каждая грань представляется списком своих вершин, причем порядок обхода вершин должен сохраняться постоянным, например, по часовой стрелке. Затем для каждой грани вычисляется нормаль. Грани, нормаль которых имеет отрицательный знак, считаются невидимыми и не отображаются.

Более простую разновидность этого метода можно использовать для фигур, которые условно считаются пересеченными плоскостью проецирования. Для

этих фигур отображаются только те грани, все вершины которых имеют неотрицательную координату Z.

3. *Метод «плавающего горизонта» (или метод Y буфера).* Метод заключается в том, что сначала изображаются ближние грани или сечения объекта, а затем лишь те части более удаленных граней, которые выступают за края верхнего и нижнего горизонтов.

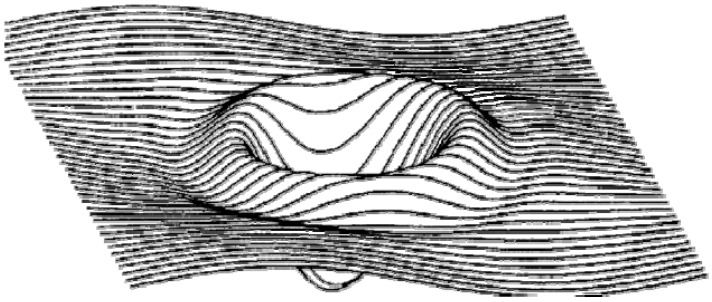


Рис. 3.10. Метод плавающего горизонта

Разработан для тех случаев, когда известны аналитические описания изображаемых объектов (например, уравнения сечений поверхности, см. рис. 3.10).

Элементы отображаются начиная с ближнего плана. *Y*-координаты сечения записываются в *Y*-буфер (одномерный массив, размер которого равен ширине изображения). В *Y*-буфере будет отслеживаться *y* максимальное. При записи следующего сечения информация в *Y*-буфере обновляется и на экран выводятся только пикселы, *y*-координаты которых больше соответствующих координат *Y*-буфера.

4. *Метод «Z-буфера»* заключается в том, что для экрана заводится двумерный массив *Z* «глубин» каждой точки экрана. Размерность массива зависит от разрешения экрана. В начале массив инициализируется значениями –∞. При выводе на экран точки с мировыми координатами (*x*, *y*, *z*) вызывается функция, которая проверяет, надо ли выводить точку (т.е. точка выводится, если ранее в позиции (*x*, *y*) не была изображена менее «глубокая» точка).

```
Function Point(x, y, z):boolean;
begin
 if Z[x,y]<z
then begin
        Z[x,y]:=z;
        Point := true (*Заслонить далекую*)
end;
 else Point := false; (*Не заслонять более близкую*)
end;
```

Для любого объекта выводятся все точки в любом порядке. Если выводимая точка имеет значение z больше, чем значение в буфере, то эту точку следует выводить, а значение z сохранить в *Z*-буфере.

Данный метод используется во многих графических 3D-акселераторах, которые аппаратно реализуют этот метод. Наиболее целесообразно, когда акселератор имеет собственную память для *Z*-буфера, доступ к которой осуществляется быстрее, чем к оперативной памяти компьютера.

3.4. Методы освещения объемных фигур

Интенсивность освещения точки, принадлежащей некоторой поверхности и освещаемой источником с интенсивностью $I_{ист}$, определяется суммой трех составляющих:

$$I = I_P + I_M + I_З .$$

Рассмотрим подробней физический смысл и математические модели каждой из составляющих освещенности.

$I_р = I_{ист} \cdot K_п$ = const – интенсивность рассеянного освещения, заполняющего пространство, где $I_{ист}$ – интенсивность источника света, $K_п$ – коэффициент прозрачности среды.

$I_м = I_{ист} \cdot K_м \cdot cos\theta$ – (закон Ламберта) интенсивность освещения матовой поверхности точечным источником. Здесь $K_м$ – коэффициент материала, θ°–°угол между нормалью к поверхности в точке и вектором на источник света (см. рис. 3.11). Нормаль – это перпендикулярный к поверхности вектор единичной длины.

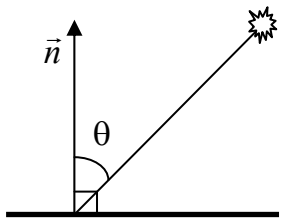


Рис. 3.11. К расчету $I_м$

$I_з = I_{ист} \cdot K_м \cdot cos^p\alpha$ – интенсивность освещения зеркально отражающей поверхности точечным источником. Здесь p – константа в интервале 1÷200, α – угол между вектором, проведенным из точки на наблюдателя, и лучом идеально отраженного света (см. рис. 3.12).

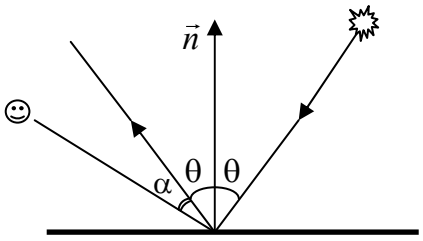


Рис. 3.12. К расчету $I_з$

Рассмотрим методику расчета $cos\theta$ (см. рис. 3.13).

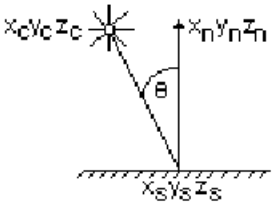


Рис. 3.13. К расчету cosθ

Выделим на плоскости произвольный треугольник, заданный тремя вершинами с координатами (x_1, y_1, z_1), (x_2, y_2, z_2) и (x_3, y_3, z_3), а положение источника света – координатами (x_c, y_c, z_c). Вместо вектора, указывающего на источник света, целесообразно рассматривать параллельный ему вектор с началом в одной из вершин треугольника, например в вершине (x_1, y_1, z_1):

$$(x_s, y_s, z_s) = (x_c\text{-}x_1, y_c\text{-}y_1, z_c\text{-}z_1).$$

Тогда координаты вектора нормали вычисляются следующим образом:

$$x_n = (y_2\text{-}y_1)(z_3\text{-}z_1)\text{-}(z_2\text{-}z_1)(y_3\text{-}y_1),$$

$$y_n = (z_2\text{-}z_1)(x_3\text{-}x_1)\text{-}(x_2\text{-}x_1)(z_3\text{-}z_1),$$

$$z_n = (x_2\text{-}x_1)(y_3\text{-}y_1)\text{-}(y_2\text{-}y_1)(x_3\text{-}x_1),$$

а косинус угла между векторами (x_s, y_s, z_s) и (x_n, y_n, z_n), с учетом определения скалярного произведения двух векторов, определяется из соотношения:

$$\cos\theta = \frac{x_n \cdot x_s + y_n \cdot y_s + z_n \cdot z_s}{\sqrt{(x_n{}^2 + y_n{}^2 + z_n{}^2)(x_s{}^2 + y_s{}^2 + z_s{}^2)}}.$$

Радиус-вектор – это вектор с начальной точкой в начале координат.

Метод Гуро. Этот метод предназначен для создания иллюзии гладкой криволинейной поверхности, описанной в виде многогранников или полигональной сетки с плоскими гранями. Если каждая плоская грань имеет один постоянный цвет, определенный с учетом отражения, то различные цвета соседних граней очень заметны, и поверхность выглядит именно как многогранник. Это происходит потому, что зрение человека имеет способность подчеркивать перепады яркости на границах смежных граней – такой эффект называется *эффектом полос Маха*. При этом увеличение числа граней приводит к существенному замедлению визуализации.

Метод Гуро основывается на идее закрашивания каждой плоской грани не одним цветом, а плавно изменяющимися оттенками, вычисляемыми путем интерполяции цветов примыкающих граней.

Рассмотрим этапы процесса закраски:

1) вычисляются нормали к каждой грани;

2) вычисляются нормали в вершинах путем усреднения нормалей примыкающих граней:

$$N=(N_1+N_2+N_3)/3;$$

3) на основе нормалей в вершинах вычисляются значения интенсивностей освещения в вершинах;

4) полигоны граней закрашиваются цветом, полученным с помощью линейной интерполяции интенсивностей в вершинах.

Интерполированная интенсив-ность I в точке (x, y) определяется исходя из пропорции (см. рис. 3.14)

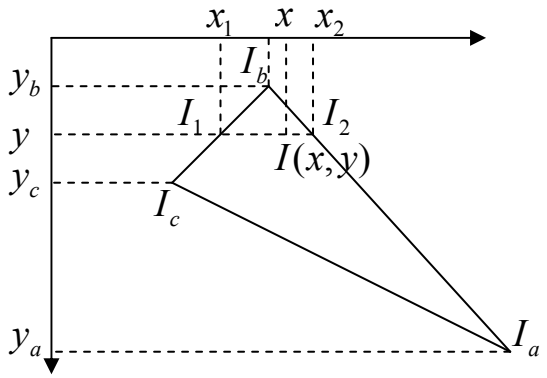


Рис. 3.14. Закраска грани методом Гуро

$$(I - I_1)/(x - x_1) = (I_2 - I_1)/(x_2 - x_1)$$

Отсюда

$$I = I_1 + (I_2 - I_1)(x - x_1)/(x_2 - x_1).$$

Значения интенсивностей и на концах горизонтального отрезка представляют собой интерполяцию интенсивности в вершинах:

$$(I_1 - I_b)/(y - y_b) = (I_c - I_b)/(y_c - y_b),$$

$$(I_2 - I_b)/(y - y_b) = (I_a - I_b)/(y_a - y_b).$$

или

$$I_1 = I_b + (I_c - I_b)(y - y_b)/(y_c - y_b),$$

$$I_2 = I_b + (I_a - I_b)(y - y_b)/(y_a - y_b).$$

Метод Фонга аналогичен методу Гуро, но при его использовании интерполируются не интенсивности отражения света, а вектора нормалей.

1) определяются нормали ко всем граням;

2) по нормалям к граням вычисляются нормали в вершинах;

3) в каждой точке закрашиваемой грани определяется интерполируемый вектор нормали;

4) по направлению вектора нормали для каждой точки определяется ее интенсивность.

Рассмотрим, как получить вектор нормали в каждой точке грани (см. рис. 3.15). Для интерполяции будем оперировать векторами N'_a, N'_b и N'_c, исходящими из центра координат плоскости проецирования и параллельными соответствующим нормалям N_a, N_b и N_c, в вершинах a, b и c.

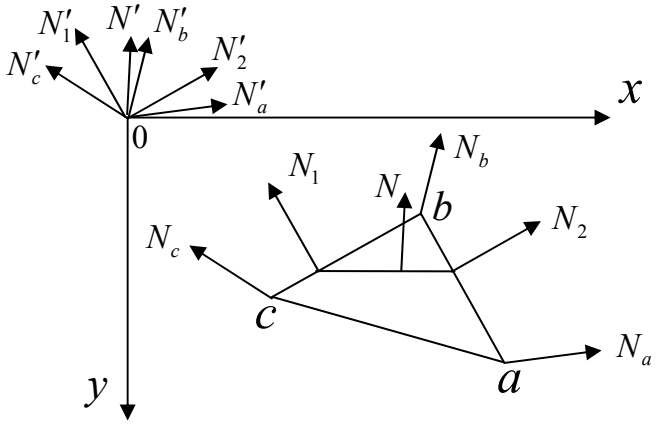


Рис. 3.15. Интерполяция векторов нормали

Координаты векторов N'_1, N'_2 вычисляются следующим образом:

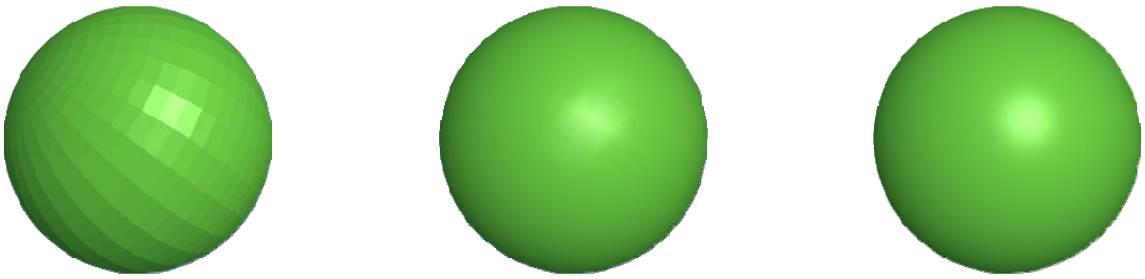
$$N'_1 = \begin{bmatrix} x_{N1} \\ y_{N1} \\ z_{N1} \end{bmatrix} = \begin{bmatrix} x_{Nb} + (x_{Nc} - x_{Nb})(y - y_b)/(y_c - y_b) \\ y_{Nb} + (y_{Nc} - y_{Nb})(y - y_b)/(y_c - y_b) \\ z_{Nb} + (z_{Nc} - z_{Nb})(y - y_b)/(y_c - y_b) \end{bmatrix},$$

$$N'_2 = \begin{bmatrix} x_{N2} \\ y_{N2} \\ z_{N2} \end{bmatrix} = \begin{bmatrix} x_{Nb} + (x_{Na} - x_{Nb})(y - y_b)/(y_a - y_b) \\ y_{Nb} + (y_{Na} - y_{Nb})(y - y_b)/(y_a - y_b) \\ z_{Nb} + (z_{Na} - z_{Nb})(y - y_b)/(y_a - y_b) \end{bmatrix},$$

где x_{Na}, y_{Na}, z_{Na}, x_{Nb}, y_{Nb}, z_{Nb}, x_{Nc}, y_{Nc}, z_{Nc} – координаты векторов N'_a, N'_b и N'_c Координаты вектора N' вычисляются по формуле:

$$N' = \begin{bmatrix} x_N \\ y_N \\ z_N \end{bmatrix} = \begin{bmatrix} x_{N1} + (x_{N2} - x_{N1})(x - x_1) / (x_2 - x_1) \\ y_{N1} + (y_{N2} - y_{N1})(x - x_1) / (x_2 - x_1) \\ z_{N1} + (z_{N2} - z_{N1})(x - x_1) / (x_2 - x_1) \end{bmatrix}.$$

На рис. 3.16 приведено сравнение методов закраски плоских граней (однотонная закраска и градиентная).



а) плоские грани б) метод Гуро в) метод Фонга

Рис. 3. 16. Сравнение методов закраски плоских граней

Метод обратной трассировки лучей. В более сложных моделях учитывается освещенность объектов не только прямыми лучами со стороны источника света, но и отраженными лучами со стороны других освещенных объектов. Главным методом для расчета освещения таких сцен является метод обратной трассировки лучей.

Принцип обратной трассировки лучей состоит в том, что через каждую точку экрана как бы проводится обратный луч света до пересечения с ближайшим объектом сцены, далее из этой точки проводится луч в направлении источника света (см. рис. 3.17).

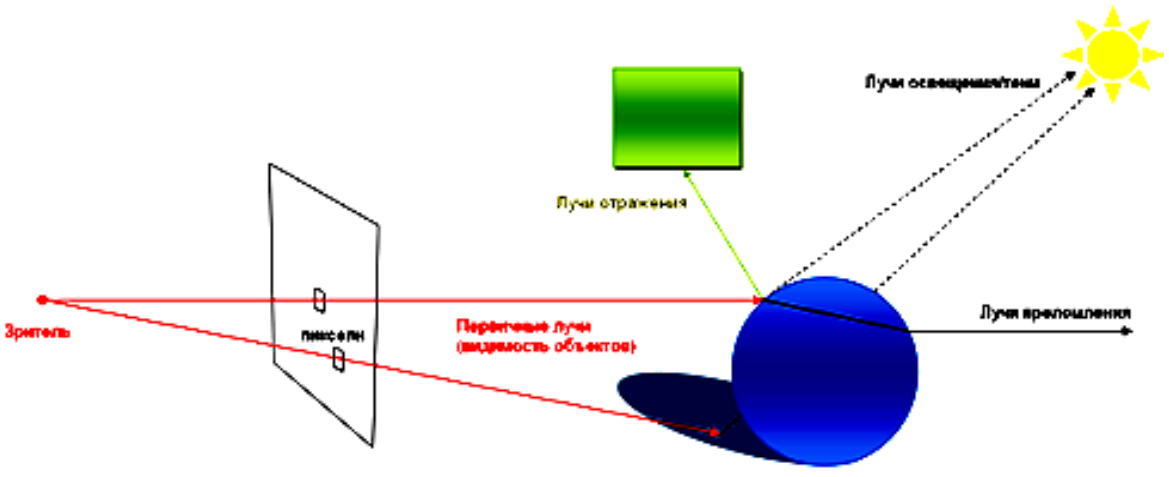


Рис. 3.17. Метод обратной трассировки лучей

Если луч, выпущенный на источник света, ничего не пересекает на своем пути, то данная точка освещена, иначе она лежит в тени. Если луч попадает на зеркальную поверхность, то, в соответствии с законами оптики, выпускается

отраженный луч, что дает возможность построить отражение. В зависимости от свойств среды, через которую проходит луч, он может преломляться, что позволяет моделировать сложные реалистичные световые эффекты. Этот метод позволяет получить не только тени от объектов, но и рассчитать вторичное освещение, когда отраженный тусклый свет попадает в непосредственно затененные области и размывает тени.

Данный метод основан на принципе обратимости луча света, который гласит, что луч, «двигаясь» в противоположном направлении, пройдет в пространстве тот же самый путь.

Метод обратной трассировки лучей очень ресурсоемок, он требует больших затрат оперативной памяти и процессорного времени. Существует большое количество сопутствующих методов, упрощающих и ускоряющих расчеты. Например, метод оболочек предусматривает построение для каждого объекта «оболочки» – простой фигуры (шара или куба), содержащей внутри себя объект. На первом этапе расчета анализируются взаимные расположения не самих объектов, а их оболочек, и только если одна оболочка загораживает другую, выполняется более подробный расчет. В противном случае фигуры считаются не загораживающими друг друга.

На рис. 3.18 приведено изображение, полученное с помощью метода обратной трассировки лучей.

Рис. 3.18. Результат работы метода обратной трассировки лучей

ГЛАВА 4. СОВРЕМЕННЫЕ ГРАФИЧЕСКИЕ СИСТЕМЫ

Применение методов и средств компьютерной графики в самых разных областях – в вычислительной технике, на телевидении и в кинематографии, в научных исследованиях и т.п. – предъявляет к ним требования унификации и стандартизации.

4.1. Классификация и обзор современных графических систем

Эволюция графических стандартов естественно отражает процесс развития средств компьютерной графики – от векторной графики до систем виртуальной реальности.

В основе разработки графических стандартов лежит принцип виртуальных ресурсов, позволяющий разделить графическую систему на несколько слоев – прикладной, базисный и аппаратно-зависимый. При этом каждый слой является виртуальным ресурсом для верхних слоев и может использовать возможности нижних слоев с помощью стандартизованных программных интерфейсов. Кроме того, графические системы могут обмениваться информацией с другими системами или подсистемами с помощью стандартизованных файлов или протоколов. В соответствии с этими соображениями первоначально были выделены три основных направления стандартизации – базисные графические системы, интерфейсы виртуального устройства, форматы обмена графическими данными.

Стандартизация базисных графических систем направлена на обеспечение мобильности прикладных программ и основана на концепции ядра, содержащего универсальный набор графических функций, общих для большинства применений.

Наиболее известными проектами по стандартизации базисных систем являются Core System, GKS, GKS-3D, PHIGS, PHIGS+. Основное направление развития этих проектов заключалось в усилении изобразительных возможностей для визуализации геометрических объектов (2D, 3D, удаление скрытых линий и граней, полутоновая закраска, текстурирование и пр.). Стандарт на базисную графическую систему включает в себя функциональное описание и спецификации графических функций для различных языков программирования.

Концепция виртуального устройства начала разрабатываться с момента появления аппаратно-независимых графических систем. Интерфейс виртуального устройства разделяет аппаратно-зависимую и аппаратно-независимую части графической системы. Он обеспечивает заменяемость графических устройств (терминальную независимость), а также возможность работы с несколькими устройствами одновременно. Интерфейс виртуального устройства может существовать в форме программного интерфейса и/или протокола взаимодействия двух частей графической системы. Наиболее четко концепция виртуального устройства представлена в проекте CGI.

Развитие этой концепции совпало с активным перемещением графических средств на персональные компьютеры и графические станции. При этом основными интерактивными устройствами стали растровые дисплеи, а устройствами для получения твердых копий – растровые принтеры. Это привело к необходимости выделения отдельного набора растровых функций, позволяющих использовать функциональные возможности растровых устройств.

Дальнейшее развитие растровых функций связано с появлением многооконных графических систем X Window и MS Windows (а также NeWS и Display Postscript), обеспечивших удобные средства для манипулирования растровыми изображениями. Эти средства явились основой для развития систем обработки изображений и для организации эффективного многооконного пользовательского интерфейса с использованием меню, диалоговых панелей, полос просмотра и пр. Отметим, что традиционные средства вывода геометрических примитивов (линий, дуг, многоугольников) и текстов также имеются в этих системах.

Сегодня наиболее развитые проекты PEX и OpenGL неплохо совмещают основные достижения как геометрического, так и растрового направления.

4.2. Основные функциональные возможности современных графических систем

4.2.1. Графические системы класса 2D

GKS – стандарт ISO на базисную графическую систему. Впервые опубликован в 1982 году. Принят в качестве международного стандарта в 1985 году. Разработаны спецификации GKS для языков C, Fortran, Pascal, Ada. В соответствии или с учетом стандарта GKS разработано большое количество графических систем, например GKS-3D и PHIGS.

Функции управления обеспечивают работу с несколькими логическими рабочими станциями ввода/вывода. Одной из категорий рабочих станций является метафайл. Поддерживается таблица состояния системы, а также таблицы конфигурации и состояния рабочих станций. Имеется более 100 функций опроса возможностей и текущего состояния системы.

Функции вывода поддерживают шесть примитивов – ломаная линия, набор маркеров, заполненная область, текст, массив ячеек и обобщенный графический примитив. Более 30 функций управления атрибутами (линий, маркеров, заполнения и текста) обеспечивают индивидуальное изменение атрибутов и объединение их в группы, связанные с рабочими станциями. Преобразование координат двухступенчатое – нормализация и преобразование рабочей станции.

Поддерживается сегментация. Атрибуты сегментов – видимость, указуемость, выделенность, приоритет, преобразование. Сегменты могут копироваться на рабочую станцию, удаляться, включаться в другие сегменты.

Растровые функции отсутствуют. Используемая цветовая модель – индексированная таблица RGB (Red-Green-Blue).

Функции ввода поддерживают логические устройства ввода координат, линий, чисел, текстовых строк, а также устройства выбора и указания. Устройства ввода могут работать в режимах запроса, опроса и обработки событий.

MGKS или MiniGKS – сокращенные варианты GKS без сегментации и с минимальным количеством функций опроса. Эти проекты прошли мимо внимания разработчиков стандартов, но были поддержаны многими разработчиками конкретных графических систем.

GKS-N, или New GKS, – проект, обсуждавшийся в ISO (1989 год), направлен на улучшение функциональных характеристик GKS. Заметно явное влияние проекта CGI. Последующих публикаций не было.

PostScript (Adobe Systems, 1985) – язык описания страниц для растровых печатающих устройств. Отличительная особенность – широкие изобразительные возможности при минимальном наборе графических функций. Множество графических систем и настольных издательских систем поддерживают PostScript. Некоторые производители лазерных принтеров обеспечивают его аппаратную поддержку. PostScript использован для выполнения графических функций в многооконных системах NeWS и Display PostScript. Привлекательные свойства этого языка способствовали появлению его трехмерных расширений.

Широкие изобразительные возможности языка PostScript обеспечены понятием траектории (path), которая может быть составлена из линий, дуг, сегментов кривой Безье и текстовых символов. В процессе вывода траектории могут подвергаться произвольным линейным преобразованиям. Замкнутые траектории могут быть закрашены, заполнены растровым образцом (pixmap) или заштрихованы другими траекториями. Заполнение может производиться по различным правилам (even-ocЬI, nonzero-winding-number). Линии могут быть различного типа, переменной толщины и иметь скругления в точках соединения. Работа с текстами происходит на основе богатой библиотеки шрифтов. Поддерживается несколько цветовых моделей – RGB, CMY (CyanMagenta- Yellow) и HSV (Hue-Saturation-Value).

CGI – проект стандарта (ISO, 1986) на интерфейс виртуального устройства. На стадии обсуждения этот проект фигурировал в публикациях под названием VDI. CGI ориентирован не на прикладных, а на системных программистов, занимающихся разработкой графических систем. Функциональные возможности CGI сформированы с учетом разработанных ранее проектов GKS и CGM (Computer Graphics Metafile). Заметно влияние проектов PostScript и X Window System.

Функции вывода поддерживают работу с линиями, многоугольниками, прямоугольниками, маркерами, текстами, дугами, секторами и сегментами круга и эллипса, а также замкнутыми фигурами, составленными из этих примити-

вов. Замкнутые объекты могут закрашиваться, заштриховываться или заполняться растровым образцом. Набор атрибутов CGI аналогичен набору атрибутов GKS. Конвейер преобразования ограничен преобразованием рабочей станции.

Функции сегментации аналогичны имеющимся в GKS.

Растровые функции поддерживают работу с отображаемыми и виртуальными битовыми картами. Первые являются частью видеопамяти устройства. Вторые могут быть полноцветными или двухцветными матрицами пикселов в неотображаемой памяти. Двухцветные виртуальные битовые карты могут служить в качестве маски для операции заполнения областей, а также для задания символов, маркеров, курсоров и пр. Атрибутами карт являются прозрачность, основной и фоновый цвет. Введены различные режимы наложения цветов при выводе пикселов (and, or, xor, ...).

Функции ввода аналогичны имеющимся в GKS с некоторыми дополнениями. Введено понятие триггера, позволяющего установить режим срабатывания отдельных устройств в зависимости от некоторого события. Более четко определены понятия подсказки, эха и подтверждения. Введены два новых логических устройства ввода – растровая область и обобщенное устройство ввода.

X Window System – многооконная графическая система, разработанная в Массачусетском Технологическом институте. Первые публикации появились в 1986 году. Одна из основных целей разработки – обеспечение сетевой прозрачности и возможности использования широкого спектра цветных и монохромных графических станций.

Система разделена на две части – клиент и сервер – взаимодействующие с помощью X-протокола. Прикладному программисту предоставлена библиотека базисных функций X Lib и надстроенная над ней библиотека инструментальных средств X Toolkit. Функции управления обеспечивают возможность манипулирования системой окон и контроля за действиями пользователя. Параметры графических функций включают в себя идентификаторы дисплея и окна, а также графический контекст, содержащий значения атрибутов и другие параметры отображения.

Функции вывода обеспечивают изображение точек, линий, дуг, окружностей, прямоугольников, а также заполнение многоугольников, секторов, сегментов и прямоугольников. Аналогично языку PostScript имеются атрибуты, определяющие способ скругления ломаных линий и правило заполнения. Функции вывода текстов поддерживаются богатой библиотекой шрифтов. Конвейер преобразования координат отсутствует.

Структуризация или сегментация данных не поддерживается.

Растровые функции обеспечивают широкие возможности для манипулирования с битовыми и пиксельными матрицами (Bitmap, Pixmap). Кроме того, пиксельные матрицы могут использоваться в качестве образца заполнения, а битовые – в качестве маски отсечения. Используемая цветовая модель – RGB.

Функции ввода на базисном уровне обеспечивают развитый механизм обработки событий, от мыши и клавиатуры. Функции более высокого уровня (X Toolkit и библиотека виджетов) обеспечивают работу с меню, диалоговыми панелями, полосами просмотра и пр.

Microsoft Windows – многооконная надстройка над операционной системой MS-DOS на IBM PC. Версия Windows NT трансформировалась в полноценную операционную систему. Обеспечивает многозадачный режим. Графические функции системы аналогичны имеющимся в X Window, однако в параметрах функций нет идентификатора дисплея. Поддерживается метафайл.

NeWs (Sun Microsystems, 1987) и Display Postscript (Adobe Systems,1990) – многооконные графические системы, в основе которых лежит PostScript. Обладают эффективными графическими возможностями, унаследованными от языка PostScript. В системе NeWS появились 3D траектории.

4.2.2. Графические системы класса 3D

Core System – первый проект (ANSI) по стандартизации базисной графической системы. Функциональное описание было опубликовано в 1977 году. На этот проект были замкнуты усилия многих разработчиков графических средств в течение последующих 5 лет. Построен на концепции рисующего элемента (2D и 3D) и обеспечивает работу только с линиями, маркерами и текстами. Для управления параметрами проектирования используется аналогия с камерой. Поддерживается сегментация. После появления стандартов GKS-3D и PHIGS проект Core System потерял свою актуальность.

GKS-3D – расширенный вариант GKS (ISO, 1987), позволяющий работать с трехмерными графическими объектами. В этот проект включены следующие дополнительные (по отношению к GKS) возможности:

1) Функции вывода дополнены семью 3D-примитивами – те же, что в GKS, с приставкой 3D и набор заполняемых областей 3D. Для последнего примитива введены атрибуты контура, аналогичные атрибутам линий. Введен атрибут для управления алгоритмами удаления скрытых линий и граней. Введены 3D-преобразование, 3D-нормализация, видовое преобразование, 3D-преобразование рабочей станции. Видовое преобразование позволяет производить параллельное и центральное проецирование.

2) Функции сегментации расширены возможностью работы с 3D-сегментами. Введено преобразование 3D-сегментов.

3) Функции ввода дополнены двумя логическими устройствами для ввода координат 3D и линий 3D.

XGKS, GEX – проекты объединения систем X Window и GKS/GKS-3D. Обсуждались в литературе по стандартизации, но не получили дальнейшего развития.

PHIGS – альтернативный по отношению к GKS-3D стандарт (ANSI-1986, ISO-1989), обеспечивающий возможность интерактивных манипуляций с иерархически структурированными графическими объектами. Получил дальнейшее развитие в проектах PHIGS+ и PEX. Сравнительные с GKS-3D характеристики следующие: набор примитивов и атрибутов аналогичен имеющимся в GKS-3D. Поддерживается несколько цветовых моделей – RGB, CIE (Commission Internationale de l'Eclairage), HSV (Hue-Saturation-Value), HLS (Hue-Lightness-Saturation). Вместо 3D преобразования нормализации введено модельное преобразование. Вместо сегментов введены иерархические структуры данных. Структуры могут включать в себя примитивы, атрибуты, преобразования, неграфические данные, а также ссылки на другие структуры. Средства редактирования позволяют удалять и копировать элементы структур. Включен механизм фильтрации, осуществляющий выборочное отображение элементов, их выделение и пр.

PHIGS+(или PHIGS-PLUS) – проект расширения PHIGS (ISO/ANSI Draft 1990), направленный на обеспечение основных требований прикладных программ в области освещения, полутоновой закраски и эффективного описания сложных поверхностей. Для этих целей в PHIGS+ включен следующий набор примитивов:

- набор полилиний с данными,
- кривая нерационального B-сплайна,
- кривая нерационального B-сплайна с данными,
- полигональная область с данными, набор полигональных областей с данными,
- набор треугольников с данными,
- полоса треугольников с данными, набор четырехугольных ячеек с данными,
- поверхность нерационального B-сплайна,
- поверхность нерационального B-сплайна с данными.

Примитивы, имеющие суффикс «с данными» позволяют включить дополнительную информацию, являющуюся частью определения примитива. Например, в случае набора треугольников для каждой грани и/или вершины можно задать комбинации цвета, нормаль и прикладные данные. Далее, существует механизм управления, позволяющий определить, какие данные следует использовать, а какие пропустить во время отображения. PHIGS+ различает переднюю и заднюю поверхности грани на основе геометрической нормали. Различные значения цвета и другие атрибуты могут быть определены для передней и задней граней. Для вычисления освещенности, кроме геометрических характеристик, задаются отражательные свойства поверхности, а также расположение источников цвета и их характеристики.

PEX (MIT X Consortium) – проект расширения системы X Window для поддержки PHIGS+. Первоначальная версия XPHIGS 1.0 – 1987 год, последняя версия PEX 6.0 – 1992 год. Одна из двух систем (другая – OpenGL), обеспечивающих наиболее развитые на сегодняшний день инструментальные средства для построения реалистичных изображений. Суть проекта PEX состоит в описании механизма расширения X-протокола и X-сервера для обеспечения функций PHIGS+, что, в первую очередь, предназначено для системных программистов. С точки зрения прикладного программиста функциональные возможности PEX в части изображения пространственных объектов соответствуют системе PHIGS+. Однако начиная с версии 5.2 в PEX появились новые возможности, обеспечивающие устранение ступенчатости (antialiasing) и текстурирование поверхностей. Средства работы с растровыми изображениями поддерживаются с помощью X Window и дополнительных расширений.

4.3. Организация диалога в графических системах

Начало интерактивных вычислений и, следовательно, исследование человеко-машинного интерфейса принято отсчитывать с 1959 г., когда на конференции Юнеско по обработке информации Г. Стречи предложил режим разделения времени при решении задач на компьютерах. По мере роста мощности компьютеров росли и затраты на диалоговую компоненту программного обеспечения. Вопрос эффективности использования машин обострился во время стремительного выхода на рынок рабочих станций, объединивших интерактивность с графикой. Термин эффективность с тех пор изменил свое значение. Если раньше он отражал такие характеристики, как процессорное время и объем занимаемой памяти, то теперь под ним понимают простоту разработки, легкость сопровождения и удобство работы с программой. Поэтому затраты на исследование и разработку пользовательского интерфейса являются оправданными. В настоящее время большие усилия прикладываются к разработке методов и созданию инструментальных средств в рамках систем, получивших название UIMS – User Interface Management System.

Традиционный графический подход к интерфейсу с пользователем связан с работами Сазерленда, Ньюмена и др., в которых взаимодействие базируется на использовании графического дисплея с регенерацией и светового пера. Дальнейшее развитие графического диалога связано с прогрессом в области систем интерактивной машинной графики, который привел к регламентации в виде международных стандартов.

Все основные черты интерфейса с пользователем на современных рабочих станциях суть производные от работ по Xerox Park:

- управление окнами;
- использование графических символов («икон») для представления объектов;

- стиль взаимодействия, называемый непосредственным манипулированием;
- популярность мыши как устройства позиционирования на экране;
- объектно-ориентированный стиль программирования.

С тех пор система классификации инструментария для создания и управления пользовательским интерфейсом рассматривается на трех уровнях:

- системы управления окнами (WMS-Window Manager System);
- специализированный инструментарий;
  - обычный (MacIntosh, SunView);
  - объектно-ориентированный (Smalltalk-80, Andrew, InterView);
- системы управления пользовательским интерфейсом.

Многооконная технология обеспечивает пользователя доступом к бóльшему объему информации, чем это возможно при работе с одним экраном. Окна дают доступ ко множеству источников информации. Пользователь может объединять информацию от нескольких источников, исследовать информацию на разных уровнях детализации. В мультипрограммном режиме есть возможность управлять несколькими параллельными задачами. Вход и выход каждой задачи отображается в разных окнах, позволяя пользователю сосредоточиться по необходимости на каждой задаче.

Интерфейс со стороны оператора и прикладной программы содержит команды заведения/уничтожения окон, изменения их размеров и положения, поднятия наверх, сжатия окна до пиктограммы и восстановления. Содержит графическую библиотеку вывода (только основные примитивы) и обработчик событий. Другими словами есть некие механизмы для реализации пользовательского интерфейса.

В научной литературе пока нет согласованного взгляда на термин UIMS – точное его значение само является объектом исследования. Одна из версий принадлежит Майерсу: «Система проектирования интерфейса пользователя есть интегрированный набор средств, помогающих программисту в создании и управлении различными интерфейсами пользователя. Эти системы обычно называют системами управления пользовательским интерфейсом (UIMS – User Interface Management Systems), но предпочтительнее называть их системами проектирования (UIDS – User Interface Development Systems), поскольку UIMS ассоциируется только с частью системы, работающей во время исполнения программы (но не с частью, используемой во время разработки), или с системами, включающими явные компоненты управления диалогом. UIDS обеспечивает как разработку, так и реализацию интерфейса и, таким образом, покрывает более широкий класс программ».

Основной концепцией UIDS является идея строгого разделения интерфейса и прикладной программы. В идеале она должна поддерживать все стили диалога и упрощать построение сложных интерфейсов. UIDS должен обеспечивать

язык определения интерфейса для представления требуемого диалога и генератор, которой автоматически создает необходимый код из исходного определения в этом языке. Эти функции во многом похожи на функции компилятора или интерпретатора для обычных языков программирования.

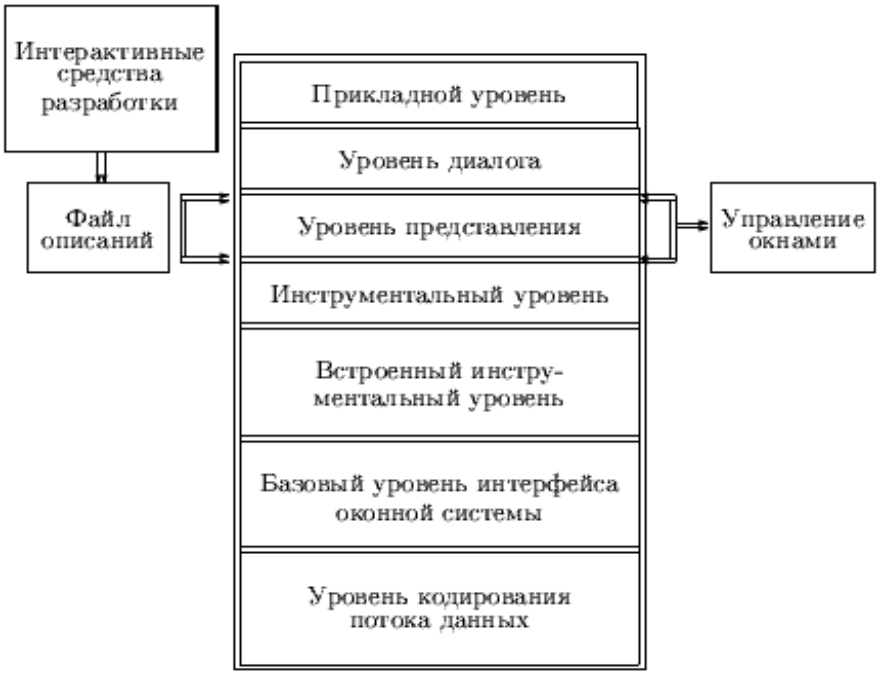


Рис. 4.1. Компоненты UINS/UIDS

Структурная схема компонент UIMS/UIDS представлена на рис. 4.1.

4.4. Операционная система MS Windows

Операционная система Windows предоставляет программистам возможность использовать в своих программах сотни разнообразных функций, доступ к которым сделан в виде интерфейса API (Application Program Interface – интерфейс прикладного программирования). В ранних 16-разрядных версиях Windows существовало четкое разделение Win API на три части:

- KERNEL – отвечала за базовые средства операционной системы, включающие работу с файлами, с памятью, запуск и завершение процессов и т.п.;
- USER – отвечала за работу с клавиатурой, мышью, параллельным и последовательным портами и т.п.;
- GDI – отвечала за графический интерфейс.

В более современных 32-разрядных версиях структурное разделение исчезло, осталось только функциональное разделение API на три подсистемы.

Основной недостаток стандартных средств отображения графики в Win API – невысокая скорость вывода изображений на экран. Для ускорения этого процесса нередко построение изображения выполняют с использованием контекста оперативной памяти, а затем быстро копируют его в окно.

В общем, быстродействия базовых графических средств Win API вполне хватает для офисных приложений и прочих программ, не использующих динамичной смены изображений на экране.

При разработке программы на основе компьютерных языков, таких как C, C++, Pascal, в текст программы можно включать вызовы функций, которые входят в состав API Windows. После компиляции создается выполняемый файл (*.exe), который можно запускать на разных компьютерах с различными версиями ОС Windows, и программа будет корректно выполняться. Поскольку сами функции располагаются в модулях операционной системы, а в программе содержатся только вызовы функций (call), то код выполняемого файла имеет небольшой размер.

4.4.1. Объектная архитектура Windows

Несмотря на то, что в среде Windows могут функционировать программы, не использующие оконного ввода-вывода, тем не менее они составляют незначительное меньшинство от всех Windows-программ. Типичное приложение Windows функционирует в рамках возможностей, предоставляемых графическим пользовательским интерфейсом.

Графический интерфейс Windows построен по *объектной технологии*. В операционной системе используется много различных объектов, каждый из которых принадлежит какому-то классу, но один из них играет ключевую роль, это – «окно». С точки зрения пользователя, *окно* – это прямоугольная область экрана, с которой ассоциирована некоторая Windows-программа. Одной программе может быть поставлено в соответствие несколько окон, но одно из них всегда считается «главным». На экране могут одновременно располагаться окна нескольких приложений, одно из них является «активным».

Рассмотрим принципы организации диалога в Windows. С каждым окном может быть связано некоторое количество компонентов диалогового взаимодействия – меню, кнопок, полей ввода и т.п. В каждый определенный момент времени актуальными являются диалоговые компоненты только «активного» окна. Весь клавиатурный ввод направляется в окно, обладающее «*фокусом ввода*». Напротив, мышь воздействует на диалоговые компоненты того окна, на котором располагается ее курсор.

Все объекты Windows в процессе работы обмениваются сообщениями. Обмениваться сообщениями могут даже отдельные части одного приложения, например, главная программа и ее окно. Обмен сообщениями возможен как в синхронном, так и в асинхронном режиме. Сообщение – это совокупность:

- номера, идентифицирующего тип сообщения;
- набора данных, связанных с сообщением.

В Windows определены свыше тысячи стандартных сообщений. Сообщения начинаются с префикса «WM_» (Windows Message). Например: WM_CREATE посылается окну при его создании; WM_COMMAND посылается

окну при операциях с каким-нибудь диалоговым элементом этого окна (например, с меню или кнопкой); WM_SETFOCUS посылается окну в тот момент, когда к нему переходит «фокус ввода»; BM_SETSTATE посылается органу управления (например, кнопке) для того, чтобы оно принудительно изменило свое состояние, и т.п.

При получении приложением сообщения WM_CREATE выполняется код, написанный в проекте Delphi как обработчик события OnCreate, а сообщению WM_PAINT соответствует событие OnPaint.

4.4.2. Общая структура Windows-приложения

Типичное Windows-приложение начинает свою работу со следующих действий (см. рис. 4.2):

- описывает и регистрирует новый класс окна, а именно: 1) указывает параметры окна; 2) указывает ресурсы (меню, изображения, пиктограммы и т.п.), связанные с окном; 3) указывает процедуру, которая будет обрабатывать сообщения, поступающие в окно;
- отображает окно на экране (послав ему сообщение WM_CREATE);
- организует цикл приема и обработки сообщений, поступающих в приложение.

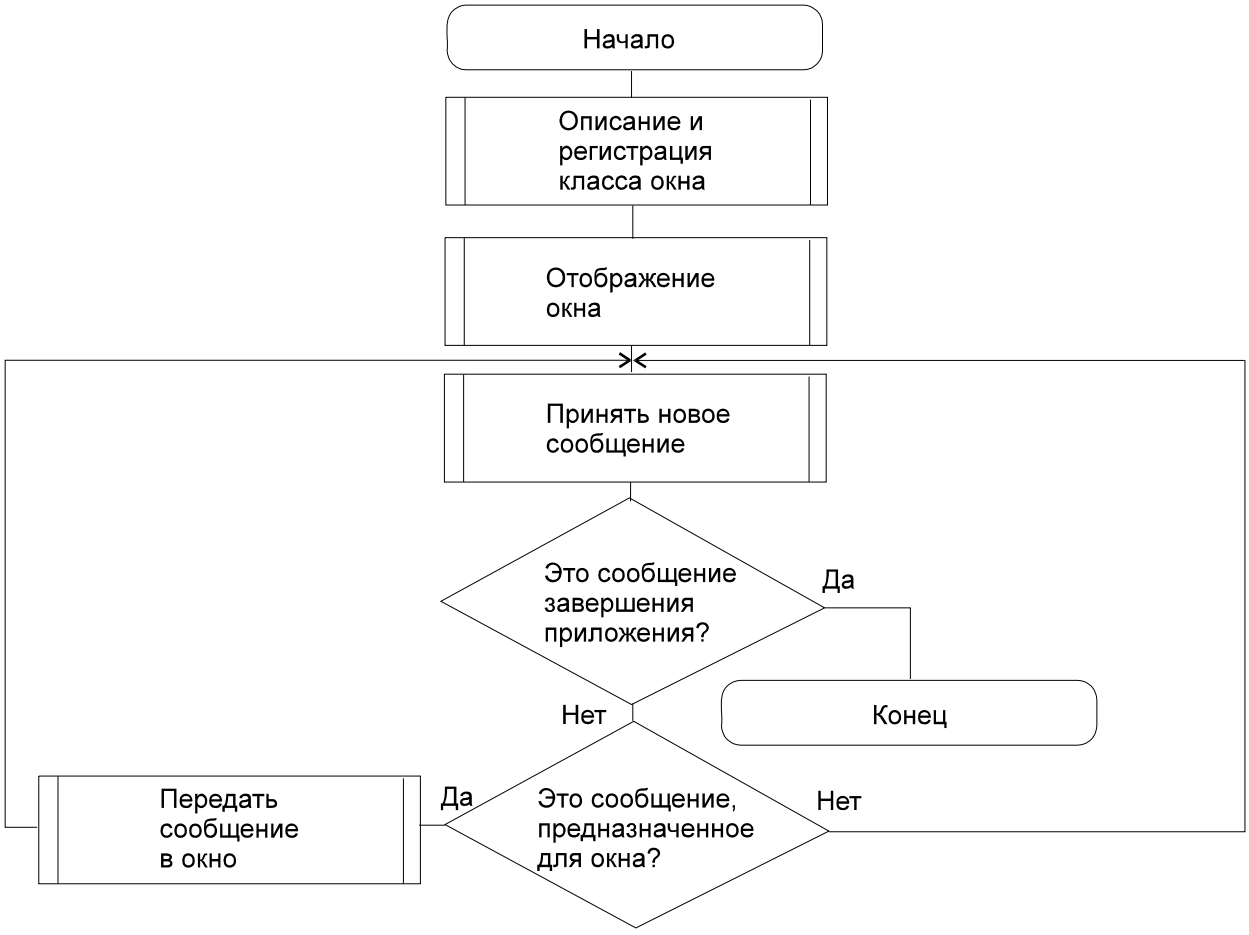


Рис. 4.2. Главная процедура Windows-приложения

Также в приложении должны быть организованы процедуры, обрабатывающие сообщения, поступающие в окно и диалоговые органы управления. Адрес процедуры обработки оконных сообщений является одним из атрибутов объекта «окно». Этот адрес используется в качестве параметра при описании и регистрации нового класса окна.

Оконная процедура может получать множество различных сообщений. Большинство сообщений могут быть обработаны автоматически. Например, обычно не требуют самостоятельной обработки сообщения, свидетельствующие о закрытии, перемещении, свертывании окна. Но для Windows-программ, производящих вывод в окно, важно обрабатывать сообщение WM_PAINT с номером 0x000F. Это сообщение автоматически передается в окно в тот момент, когда требуется перерисовка изображения (например, если окно было передвинуто или часть его была перекрыта другим окном). Целесообразно производить весь вывод изображений в окно по приходу именно этого сообщения.

4.4.3. Контекст графического устройства

Графические функции из состава API Windows объединены в отдельную подсистему GDI (Graphic Device Interface). Важная черта подсистемы GDI — аппаратная независимость многих функций от конкретного графического устройства.

С каждым физическим или виртуальным устройством, на которое Windows-программа может выводить изображение, связан контекст графического устройства (Device Context). В качестве такого устройства могут выступать: окно, принтер, файл с изображением (метафайл), область памяти. *Контекст устройства* – это структура данных, однозначно описывающая параметры устройства. Другими словами, контекст графического устройства указывает плоскость отображения, на которую делается графический вывод.

Если ваша программа делает вызов графических функций API Windows, таких как рисование точек, линий, фигур, текста и тому подобных, необходимо указывать *идентификатор контекста* (handle of device context).

Идентификатор контекста графического устройства (hdc) – это числовое значение, знание которого дает возможность направить графический вывод в нужное место. Перед началом рисования необходимо получить это числовое значение. После рисования обычно нужно освободить, деактивизировать контекст. Несоблюдение этого требования чревато неприятными последствиями — от утечек памяти до прекращения нормальной работы программы. Корректное использование контекста графического устройства осуществляется в такой последовательности:

1. Создание, активизация контекста, получение значения hdc (обычно, при помощи функции GetDC()).

2. Рисование с помощью графических функций API Windows, при этом идентификатор контекста передается в функции, выводящие изображение на устройства, в качестве параметра.

3. Уничтожение, деактивизация соответствующего контекста hdc (обычно при помощи функции ReleaseDC()).

Контекст окна на экране дисплея. Для того чтобы операционная система могла различать окна для осуществления диалога с ними, все окна при своем создании регистрируются в операционной системе и получают уникальный идентификатор. Тип этой величины в Delphi – HWND (Handle WiNDow).

Для рисования в окне программы на экране дисплея можно использовать два способа. Эти способы различаются как по особенностям получения значения hdc, так и по возможностям рисования.

Первый способ основывается на использовании пары функций GetDC и ReleaseDC.

Рассмотрим данный способ на примере. Создадим новый минимальный проект Delphi. В форме Form1 разместим кнопку. Обработчик события OnClick приведем к следующему виду:

```
procedure Tform1.Button1Click(Sender: TObject);
var
  dc : HDC;                    //ссылка на окно
begin
  dc := GetDC (Handle);          //задаем значение ссылки
  Rectangle (dc, 10, 10, 110, 110); //рисуем прямоугольник
  ReleaseDC (Handle, dc);        //освобождение ссылки
  DeleteDC (dc);   //удаление ссылки, освобождение памяти
end;
```

Теперь при нажатии кнопки на поверхности окна рисуется квадрат. Изображение будет оставаться на окне до тех пор, пока что-либо не загородит его либо пока форма не будет минимизирована, а затем снова развернута. Исчезновение квадрата после такой операции объясняется тем, что за перерисовку окна отвечает его собственный обработчик.

Функция GetDC получает ссылку типа HDC на устройство вывода, аргументом функци GetDC является величина типа HWND, в данном примере это Handle – свойство формы, которое является ссылкой на окно. Использование контекста графического вывода завершается вызовом функции ReleaseDC. Другую функцию для освобождения контекста в этом случае использовать не следует. Вообще этот способ можно рекомендовать везде, за исключением рисования во время обработки сообщения WM_PAINT.

Второй способ. Используется исключительно в теле обработчика сообщения WM_PAINT оконной функции.

Как правило, перехватчики сообщений для повышения надежности работы приложения описываются в блоке protected.

```
protected
   procedure WMPaint(var Msg: TWMPaint); message WM_PAINT;
…
{обработка сообщения wm_Paint}
procedure TForm1.WMPaint(var Msg: TWMPaint);
var
  ps : TPaintStruct;
  dc : HDC;
begin
  dc:=BeginPaint(Handle, ps);
  Rectangle (dc, 10, 10, 100, 100);
  EndPaint(Handle, ps);
end;
```

Строки *BeginPaint* и *EndPaint* присутствуют для более корректной работы приложения, при их удалении появляется неприятное мерцание при изменении размеров окна.

В данном примере ссылка на контекст устройства HDC была получена при помощи функции *BeginPaint*. Однако в данном случае ссылку на контекст устройства, соответствующего главному окну программы, можно было бы получить, используя свойство *Canvas.Handle* формы. Тогда соответствующий кусок кода выглядел бы следующим образом:

```
BeginPaint(Handle, ps);
Rectangle (Canvas.Handle, 10, 10, 100, 100);
EndPaint(Handle, ps);
```

Когда необходимо обрабатывать сообщения WM_PAINT? Это сообщение присылается любой программе тогда, когда повреждено изображение клиентской области окна этой программы. Повреждение изображения окна случается довольно часто, например, при отображении на экране нескольких окон – когда на окно было наложено еще одно окно, а потом после закрытия последнего окна части изображения первого окна уже нет. Если в программе предусмотрена перерисовка изображения рабочей (клиентской) области окна путем программирования обработчика сообщения WM_PAINT, то это позволяет в большинстве случаев гарантировать корректное отображение окна.

4.4.4. Стандартные графические функции API Windows

В Delphi существует возможность использовать стандартные графические функции API Windows. Они хранятся в динамической библиотеке gdi32.dll. Прототипы этих функций содержатся в модуле Windows.pas. Приведем примеры некоторых графических функций.

Функции для работы с отдельными пикселами.

```
function SetPixel(DC:HDC; X,Y: Integer; Color: Cardinal):Cardinal;
function SetPixelV(DC:HDC; X,Y: Integer; Color: Cardinal): LongBool;
function GetPixel(DC:HDC; X,Y: Integer): Cardinal
```

Функция SetPixel рисует один пиксел растра. Она имеет следующие аргументы: DC – контекст, X,Y – координаты точки, Color – цвет пиксела. Аргумент Color имеет тип 4-байтного Cardinal, причем три младших байта соответствуют компонентам RGB (в диапазоне от 0 до 255 каждая), а старший байт не используется. Цвет, кодируемый типом Cardinal, удобно задавать макросом RGB(r,g,b). Функция SetPixel возвращает значение цвета пиксела.

Кроме SetPixel в API Win32 есть функция SetPixelV, работающая немного быстрее, поскольку не возвращает значения; а также предусмотрена функция для получения цвета любого пиксела растра – GetPixel. *Функции для работы с линиями и фигурами* см. в табл. 4.1.

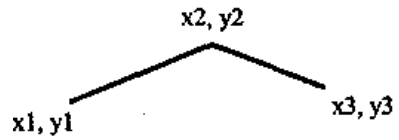
Таблица 4.1. ***Функции GDI для работы с линиями и фигурами***

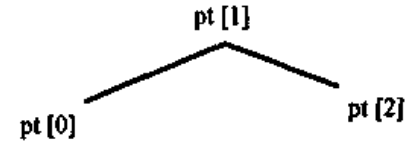
Функция	Выполняемое действие
function **AngleArc**(DC: HDC; p2, p3: Integer; p4: Cardinal; p5, p6: Single): LongBool;	Рисует дугу окружности с центром в точке (p2, p3) и радиусом p4, p5 и p6 – начальный угол и длина дуги (в градусах).
function **Arc**(DC: HDC; x1, y1, x2, y2, x3, y3, x4, y4: Integer): LongBool;	Чертит дугу эллипса в охватывающем прямоугольнике (x1,y1)-(x2,y2). Начало дуги лежит на пересечении эллипса и луча, проведенного из его центра в точку (x3,y3), а конец – на пересечении с лучом из центра в точку(x4,y4). Дуга чертится против часовой стрелки.
function **ArcTo**(DC: HDC; RLeft, RTop, RRight, RBottom: Integer; X1, Y1, X2, Y2: Integer): LongBool;	Рисует дугу эллипса, заключенного в прямоугольник. Дуга начинается в точке пересечения эллипса лучом, исходящим из центра эллипса и проходящим через точку (X1, Y1). Дуга заканчивается в точке пересечения эллипса лучом, исходящим из центра эллипса и проходящим через точку (X2, Y2).
function **Chord**(DC: HDC; X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer): LongBool;	Чертит сегмент эллипса в охватывающем прямоугольнике (x1,y1)-(x2,y2). Начало дуги лежит на пересечении эллипса и луча, проведенного из его центра в точку (x3,y3), а конец – на пересечении с лучом из центра в точку(x4,y4). Дуга чертится против часовой стрелки, а начальная и конечная точки дуги соединяются прямой.
function **Ellipse**(DC: HDC; X1, Y1, X2, Y2: Integer): LongBool;	Чертит эллипс в охватывающем прямо-угольнике (x1,y1)-(x2,y2). Заполняет внутреннее пространство эллипса текущей кистью.
function **LineTo**(DC: HDC; X, Y: Integer): LongBool;	Чертит линию от текущего положения пера до точки (x,y).
function **MoveToEx**(DC: HDC; X, Y: Integer; p4: PPoint): LongBool;	Перемещает перо в точку (X, Y) (без рисования линии). Старые координаты заносятся в переменную по указателю p4. Если p4=nil, то старые координаты не сохраняются.
function **Polyline**(DC: HDC; var Points;	Ломаная линия из многих связанных между собой

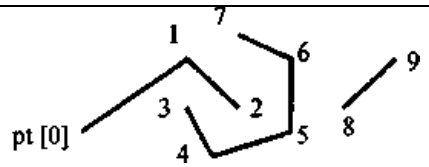
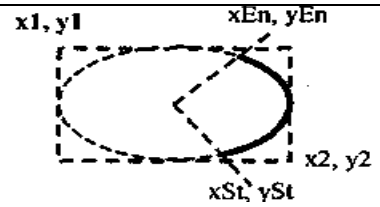
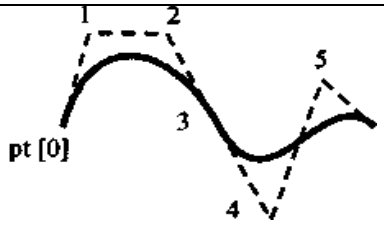
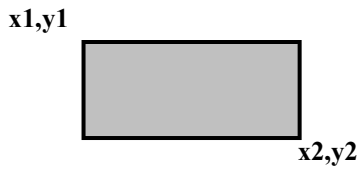
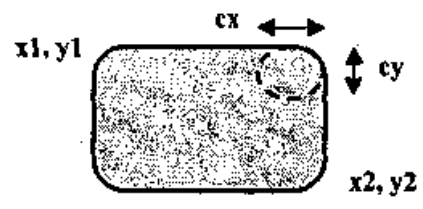
Count: Integer): LongBool;	отрезков прямых (полилиния) по точкам, заданным в массиве Points.
function **PolyLineTo**(DC: HDC; const Points; Count: Cardinal): LongBool;	Рисует ломаную линию по точкам, указанным в массиве Points. Рисование начинается из текущего положения пера. Count – количество точек.
function **PolyBezier**(DC: HDC; const Points; Count:Cardinal):LongBool;	Вычерчивает кривую Безье по опорным точкам, заданным в массиве Points.
function **PolyBezierTo** (DC: HDC; const Points; Count:Cardinal):LongBool;	Несколько связанных между собой кубических сплайнов Безье.
function **PolyDraw**(DC: HDC; const Points, Types; cCount: Integer): LongBool;	Несколько связанных отрезков прямых и сплайнов Безье.
function **Pie**(DC: HDC; X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer): LongBool;	Рисует сектор эллипса в охватывающем прямоугольнике (x1,y1)-(x2,y2). Начало дуги лежит на пересечении эллипса и луча, проведенного из его центра в точку (x3,y3), а конец – на пересечении с лучом из центра в точку(x4,y4). Дуга чертится против часовой стрелки. Начало и конец дуги соединяются прямыми с ее центром.
function **Polygon**(DC: HDC; var Points; Count: Integer): LongBool;	Вычерчивает пером многоугольник по точкам, заданным в массиве Points. Конечная точка соединяется с начальной, и многоугольник заполняется кистью. Для вычерчивания без заполнения используйте метод Polyline.
function **PolyPolygon**(DC: HDC; var Points; var PolyCount; Count: integer): LongBool;	Рисут несколько многоугольников. В массиве Points хранятся координаты всех точек. В массиве PolyCount хранится количество вершин каждого многоугольника. Count – количество многоугольников. По умолчания многоугольники не замыкаются.
function **LineDDA**(X1, Y1, X2, Y2: integer; LineFunc: TFarProc; Data: Pointer);	Вычисляет все последовательные точки в линии (X1, Y1)-(X2, Y2) и вызывает определяемую пользователем функцию LineFunc для каждой из точек, передавая ей координаты X,Y точки и Data (указатель на дополнительную информацию).
function **PolyPolyline**(DC: HDC; const PointStructs; const Points; p4: Cardinal): LongBool;	Несколько полилиний как единый объект.
function **Rectangle**(DC: HDC; X1, Y1, X2, Y2: Integer): LongBool;	Вычерчивает и заполняет прямоугольник (x1,y1)-(x2,y2). Для вычерчивания без заполнения используйте FrameRect или Poliline.
procedure **RoundeRect**(x1, y1, x2, y2, x3, y3: Integer);	Вычерчивает и заполняет прямоугольник (x1,y1)-(x2,y2) со скругленными углами, x3,y3 – радиусы скруглений.
function **TextOut**(DC: HDC; X, Y: Integer; Str: PChar; Count: Integer): LongBool;	Рисует строку текста (Str), используя заданный шрифт. Count – количество символов.

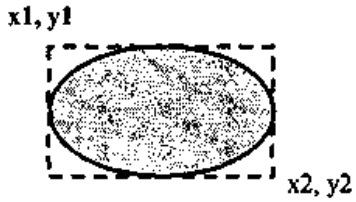
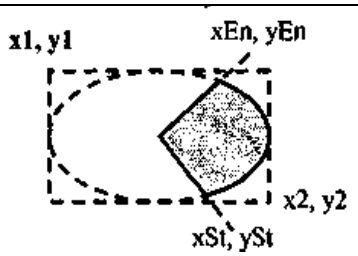
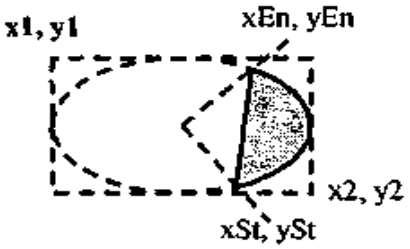
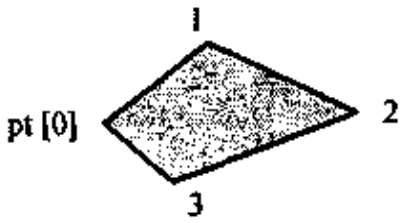
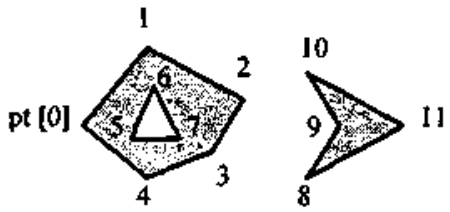
Примеры работы функций вывода графических примитивов API Windows приведены в табл. 4.2.

Т а б л и ц а 4.2. ***Примеры работы функций GDI***

Что рисуется	Программный код
	**MoveToEx**(dc,xl,yl,Nil); **LineTo**(dc,x2,y2); **LineTo**(dc,x3,y3);

	var pt: array[0..2] of TPoint;
	...
	pt[0].X:=100; pt[0].Y:=100;
	pt[1].X:=200; pt[1].Y:=50;
	pt[2].X:=300; pt[2].Y:=75;
	PolyLine(dc,pt,3);

	var pt: array [0..9] of TPoint; npt: array [1..3] of byte; ... <заполнение массива pt> npt[1]:=3; npt[2]:=5; npt[3]:=2; **PolyPolyLine**(dc,pt,npt,3);
	Arc (hdc, x1, yl, x2, y2, xSt, ySt,xEn, yEn); ... **Arc**(dc,50,50,250,150, 200,50,200,150); ...
	var pt: array [0..6] of TPoint; ... <заполнение массива pt> **PolyBezier**(dc,pt,7);
	**Rectangle**(dc, x1, y1, x2, y2);
	**RoundRect** (dc, xl, yl, x2, y2, cx, cy);

	**Ellipse** (dc, xl, yl, x2, y2);
	**Pie** (dc, xl, yl, x2, y2, xSt, ySt, xEn, yEn);
	**Chord** (dc, xl, yl, x2, y2, xSt, ySt, xEn, yEn);
	var pt: array [0..3] of TPoint; ... <заполнение массива pt> **Polygon**(dc,pt,4);
	var pt: array [0..11] of TPoint; npt: array[1..3] of byte; ... <заполнение массива pt> npt[1]:=5; npt[2]:=3; npt[3]:=4; **PolyPolygon**(dc,pt,npt,3);

Все вышеприведенные функции можно разделить на две большие группы: рисующие линии и фигуры с заполнением (закраской). Ко второй группе относятся функции: Chord, Ellipse, Pie, Polygon, PolyPolygon, Rectangle, RoundRect.

4.4.5. Стиль линии. Перо

В терминологии Windows API *перо* описывает следующие характеристики линии – цвет, толщину и стиль (пунктир). Перо – один из атрибутов контекста графического устройства. По умолчанию в контекст выбрано перо, которое соответствует черной тонкой непрерывной линии. Такое перо относится к *стандартным перьям.*

Все линии, рисуемые вашей программой с помощью функций GDI Windows API, выводятся одним и тем же цветом, имеют одинаковую толщину и тот же стиль до тех пор, пока не изменить перо. Функции рисования линий не имеют аргументов для указания текущих атрибутов линий (это характерно для гра-

фических библиотек, в которых подобные характеристики рисуемых объектов записываются как глобальные переменные, чтобы уменьшить количество аргументов вызова функций рисования).

Перо относится ко всем линиям и контурам фигур.

Чтобы начать рисовать линии, например, другим цветом, необходимо создать новое перо и выбрать его в контексте графического устройства. Отныне все линии будут рисоваться данным пером до тех пор, пока вы не выберете в контексте устройства очередное перо.

Важно то, что, создавая новое перо, нужно обязательно позаботиться об уничтожении предыдущего пера. Перо есть *объект* GDI, для него выделяется специальная область памяти. Перо существует до тех пор, пока его не уничтожить. Завершение работы прикладной программы может и не привести к автоматическому уничтожению объектов GDI и освобождению памяти компьютера. Своевременное уничтожение неиспользуемых объектов GDI возлагается на вашу программу. Иначе для некоторых версий Windows могут возникнуть утечки памяти, накопление которых может привести к нежелательным последствиям.

Не следует пытаться уничтожить стандартные перья.

Для определения стиля линий используются функции, которые имеют в своем названии Pen. Перо можно создать с помощью функции

```
function CreatePen(Style: Integer;//стиль линии
                   Width: Integer;//толщина
                   Color: Cardinal//цвет): HPEN;
```

Типичная последовательность для вывода линии с заданным стилем может быть такой:

```
var hp, hpOld: HPEN;
...
hp:=CreatePen(PS_DASHDOT, 1, RGB(255,0,0));
hpOld:=SelectObject(DC, hp); // сохраняем старое перо и выбираем наше
...// сдесь рисуем линии
SelectObject(DC, hpOld);          // выбираем предыдущее перо
DeleteObject(hp);                 // уничтожаем наше перо
```

Здесь создается перо, которое соответствует тонкой красной штрихпунктирной линии. После использования перо уничтожается.

4.4.6. Стиль заполнения. Кисть

По умолчанию в контексте графического устройства устанавливается стиль заполнения сплошным белым цветом. Для того чтобы рисовать определенную фигуру другим стилем, необходимо создать соответствующую кисть. Кисть и стиль заполнения – синонимы в API Windows.

Кисть – это объект GDI. Он требует памяти. Кроме того, все кисти, созданные во время работы программы, необходимо уничтожить, иначе они могут остаться в памяти после завершения программы. Общая схема использования кистей такая же, как и для перьев:

- создание кисти, выбор ее в контекст;
- рисование фигур с заполнением;
- освобождение контекста, уничтожение кисти.

Сплошная кисть создается функцией

```
function CreateSolidBrush(Color: Cardinal): HBRUSH;
```

Рассмотрим пример использования оранжевой кисти.

```
var hbr, hbrOld : HBRUSH;
...
    hbr:=CreateSolidBrush(RGB(255,128,0));
    hbrOld:=SelectObject(DC, hbr);// сохраняем старую кисть и
    выбираем нашу
    // здесь рисуем фигуры
    SelectObject(DC, hbrOld);      // выбираем предыдущую кисть
    DeleteObject(hbr);                    // уничтожаем нашу кисть
```

Штриховая кисть создается функцией CreateHatchBrash.

```
function CreateHatchBrush(Style: Cardinal; Color: Cardinal): HBRUSH;
```

Кисть с заданием растрового шаблона.

```
function CreatePatternBrush(Bitmap: HBITMAP): HBRUSH;

const wbits: array[1..8] of byte = (255,12,12,12,255,192,192,192);
var hbr: HBRUSH;
   bmp: HBITMAP;
...
bmp:=CreateBitmap(8,8,1,1,@wbits);
hbr:=CreatePatternBrush(bmp);
```

4.5. Спецификация HTML

Спецификация HTML представляет собой способ организации визуальных данных, передаваемых по глобальной сети Интернет.

Язык HTML (Hypertext Markup Language – язык гипертекстовой разметки) ориентирован, прежде всего, на описание текстовых страниц, возможно, содержащих некоторое количество графических изображений. Графические элементы создаются и хранятся отдельно, а в тексте на языке HTML указываются лишь позиции на экране, в которых они должны быть отображены. На рис. 4.3 приведен пример простого описания на языке HTML и внешний вид страницы, которая сформировалась в результате интерпретации этого описания.

```
<html>
<body>
<center>
<img src="zhuk.gif"></img>
<p>
Колорадский
<font size="+2" face="Arial">жук</font>
- главный враг картофельных полей!
</p>
</center>
</body>
</html>
```

а) исходное описание

б) внешний вид

Рис. 4.3. Пример HTML-страницы

Специальные программы просмотра HTML-документов, которые часто называют *браузерами*, служат для интерпретации файлов, размеченных по правилам языка HTML, форматирования их в виде Web-страниц и отображения их содержимого на экране компьютера пользователя. На сегодняшний день из всего разнообразия программ-браузеров наибольшее распространение имеют – Microsoft Internet Explorer, Mozilla Firefox, Google Chrome, Opera, Safari.

Символы, которые управляют отображением текста и при этом сами не отображаются на экране, в языке HTML принято называть *тэгами*. Все тэги языка HTML выделяются символами-ограничителями (< и >), между которыми записывается идентификатор (имя) тэга и, возможно, его параметры.

Описание документа всегда начинается с тэга <HTML>, а заканчивается тэгом </HTML>. Документ может состоять из двух разделов – раздела заголовка (начинающеюся тэгом <HEAD>) и раздела содержательной части документа (начинающеюся тэгом <BODY>).

Раздел документа HEAD определяет его заголовок и не является обязательным тэгом. Тэг-контейнер <TITLE> является единственным обязательным тэгом заголовка и служит для того, чтобы дать документу название. Оно обычно показывается в заголовке окна браузера. Название документа записывается между тэгами <TITLE> и </TITLE> и представляет собой строку текста.

Раздел документа BODY должен начинаться тэгом <BODY> и завершаться тэгом </BODY>, между которыми располагается все содержимое данного раздела. Тэг <BODY> имеет ряд параметров, приведенных в табл. 4.3, ни один из которых не является обязательным.

Т а б л и ц а 4.3. ***Параметры тэга BODY***

Параметр	Назначение
BACKGROUND	Указывает на URL-адрес изображения, которое используется в качестве фонового
BGCOLOR	Определяет цвет фона документа
BGPROPERTIES	Если установлено значение FIXED, фоновое изображение не прокручивается
ALINK	Определяет цвет активной ссылки
LINK	Определяет цвет еще не просмотренной ссылки
VLINK	Определяет цвет уже просмотренной ссылки

Для форматирования текста используются тэги физического форматирования:

- <B> – полужирный шрифт,
- <I> – курсив,
- <U> – подчеркнутый текст,
- <SUB> – нижний индекс,
- <SUP> – верхний индекс.

Тэг <FONT> позволяет указать следующие параметры шрифта:

- FACE – тип шрифта;
- SIZE – размер шрифта в условных единицах от 1 до 7;
- COLOR – цвет шрифта.

Пример:

```
<FONT FACE="Monotype Corsiva", "Arial"
COLOR=  #0000FF SIZE=3>машинная графика</FONT>
```

Тэг абзаца <P> не обязательно требует закрывающего тэга </P> . Он имеет параметр ALIGN, который может приминать значения:

- LEFT – выравнивание по левой границе окна;
- CENTER– выравнивание по центру;
- RIGHT– выравнивание по правой границе окна;
- JUSTIFY– выравнивание по ширине.

Тэг принудительного перевода строки
  не имеет закрывающего тэга.

Для разметки заголовков используются тэги <H1> … <H6>, которые могут задаваться с параметром горизонтального выравнивания ALIGN.

Тэг <HR> позволяет провести рельефную горизонтальную линию в окне, не имеет закрывающего тэга.

Текст, размеченный тэгом <PRE>, будет отображаться в таком же виде, как он отображается в обычном текстовом редакторе. Одним из важных его применений является вывод на экран больших блоков программного кода.

Тэг-контейнер <CENTER> предназначен для горизонтального выравнивания всех элементов посередине окна просмотра.

Специальные символы могут использоваться путем записи, приведенной в табл. 4.4.

Т а б л и ц а 4.4. ***Специальные символы***

Запись	Символ
&lt	<
&gt	>
	Неразрывный пробел
&copy	Знак copyright
&amp	Амперсанд
&quot	"

Комментарии в документ включаются с помощью тэга <!--комментарий-->. Все, что заключено внутри тэга, при просмотре не будет отображаться на экране.

Тэги оформления вставляемых рисунков. Для встраивания изображений в HTML-документ следует использовать тэг <IMJ>, имеющий единственный обязательный параметр SRC, определяющий URL-адрес файла с изображением. Данный тэг имеет ряд параметров.

Параметр ALIGN тэга <IMJ> указывает расположение изображения относительно текста или других элементов страницы. Возможные значения этого параметра приведены в табл. 4.5.

Т а б л и ц а 4.5. ***Параметры выравнивания рисунков***

Параметр	Значение
TOP	Верхняя граница изображения выравнивается по самому высокому элементу текущей строки.
TEXTTOP	Верхняя граница изображения выравнивается по самому высокому текстовому элементу текущей строки.
MIDDLE	Выравнивание середины изображения по базовой (нижней) линии текущей строки.
ABSMIDDLE	Выравнивание середины изображения посередине текущей строки.
BASELINE или BOTTOM	Выравнивание нижней границы изображения по базовой линии текущей строки.
ABSBOTTOM	Выравнивание нижней границы изображения по нижней границе текущей строки.
LEFT	Изображение прижимается к левому полю окна. Текст обтекает изображение с правой стороны.
RIGHT	Изображение прижимается к правому полю окна. Текст обтекает изображение с левой стороны.

Параметры WIDTH и HEIGHT задают размер изображения при отображении в пикселах или процентах от размера окна (WIDTH=num или WIDTH=num%). Значения параметров ширины и высоты изображения могут не совпадать с истинными размерами изображения. В этом случае браузеры авто-

матически при загрузке изображений выполняют его перемасштабирование. Любой из этих параметров может быть опущен. Если задан только один из параметров, то при загрузке рисунка второй параметр будет вычисляться автоматически из условий сохранения пропорций.

Параметры HSPASE и VSPASE задают отступы от изображения, оставляемые пустыми, соответственно, по горизонтали и вертикали.

Одним из параметров тэга <IMJ> является параметр ALT, определяющий альтернативный текст. При отключенном изображении вместо него на экране появится альтернативный текст. Современные браузеры будут также отображать альтернативный текст в качестве подсказки при перемещении курсора мыши в область изображения. Пример:

```
<IMG SRC="New foto.jpg" WIDTH=300
 ALIGN=CENTER ALT="Мое фото">
```

Ссылки на другие документы и файлы состоят из двух частей: из указателя ссылки и адресной части ссылки (URL-адреса). Указатели бывают двух типов – текстовые и графические.

Пример текстового указателя ссылки:

```
<A HREF="example.html">Указатель ссылки</A>
```

Пример графического указателя ссылки:

```
<A HREF="example.html"><IMG SRC=”picture.gif”></A>
```

Тэг ссылки **<A>** имеет параметр HREF, значением которого является URL-адрес. Закрывающий тэг **</A>** обязателен.

Указатель может быть как относительным, так и абсолютным. Пример абсолютного указателя: *http://www.server.com/home/index.htm*

Если в URL-адресе не указывается полный путь к файлу, то такая ссылка является относительной. В этом случае определение местоположения файлов выполняется с учетом местоположения документа, в котором имеется такая ссылка. Например, если браузер загрузил страницу, находящуюся по адресу *http://www.mysite.com/page*, то относительный указатель */picture* подразумевает адрес *http://www.mysite.com/page/picture*, т.е. подкаталог, расположенный на той же машине.

Внутренние ссылки. Для задания внутренней ссылки сначала нужно создать указатель, определяющий место назначения. Например:

```
<A NAME=chapter_5></A>
```

При этом в качестве URL-адреса используется имя ссылки с префиксом #, говорящим о том, что это внутренняя ссылка.

```
<A HREF="#chapter_5">Глава 5</A>
```

Ссылки на другие ресурсы Интернета. Для задания ссылки на электронную почту вместо URL-адреса следует указать адрес электронной почты, предварив его словом mailto:

```
<A HREF="mailto:ivanov@mail.ru">Пишите письма</A>
```

Списки. Маркированный список можно создать, используя тэг-контейнер <UL> </UL> (UL – Unordered List – неупорядоченный список). В тэге <UL> могут быть указаны два параметра: TYPE и COMPACT.

Параметр TYPE может принимать следующие значения: disc, circle и square. Этот параметр используется для принудительного задания вида мааркеров списка (закрашенные кружочки, не закрашенные кружочки, закрашенные квадратики).

Параметр COMPACT записывается без значений для указания браузеру, что данный список следует выводить в компактном виде (с уменьшенным шрифтом или расстоянием между строчек).

Каждый элемент списка должен начинаться тэгом <LI> (LI – List Item- элемент списка). В закрывающем тэге не нуждается, хотя его наличие не возбраняется. Пример:

```
<UL TYPE=disc>
<B>Знаки зодиака:</B>
  <LI>Овен
  <LI>Телец
  <LI>Близнецы
  …
</UL>
```

В качестве маркеров списка можно использовать графические изображения. Эта возможность реализуется искусственно. Для этого можно использовать тэги абзаца <P> или принудительного перевода строки
. Пример:

```
<UL>
<B>Знаки зодиака:</B><BR>
  <IMG SCR=”Green_ball.gif”>Овен<BR>
  <IMG SCR=”Green_ball.gif”> Телец<BR>
  <IMG SCR=”Green_ball.gif”> Близнецы<BR>
…
  </UL>
```

Нумерованный список создается при помощи тэга <OL> (OL – Ordered List – упорядоченный список). Отличием от маркированных списков является то, что в нумерованном списке перед каждым его элементом автоматически проставляется порядковый номер.

В тэге **<OL>** могут быть указаны следующие параметры: TYPE, COMPACT и START. Параметр TYPE используется для задания вида нумерации списка:

- TYPE = A – задает маркеры в виде прописных латинских букв;
- TYPE = a – задает маркеры в виде строчных латинских букв;
- TYPE = I – задает маркеры в виде больших римских цифр;
- TYPE = i – задает маркеры в виде маленьких римских цифр;
- TYPE = 1 – задает маркеры в виде арабских цифр.

Параметр START тэга **<OL>** позволяет начать нумерацию списка не с единицы. В качестве значения параметра START всегда должно указываться натуральное число, вне зависимости от вида нумерации списка. Пример:

```
<OL   TYPE = A   START = 5>.
```

Таблицы начинаются тэгом <TABLE> и завершаются тэгом </TABLE>. Каждая строка начинается тэгом <TR> (Table Row) и завершается тэгом </TR>. Отдельная ячейка в строке обрамляется парой тэгов <TD> и </TD> (Table Data) или <TH> и </TH> (Table Header).

Тэг <TH> используется обычно для ячеек-заголовков таблицы, а <TD> – для ячеек-данных.

В тэге <TABLE> могут использоваться следующие параметры: BORDER, CELLSPACING, CELLPADDING, WIDTH, ALIGN, HEIGHT и BGCOLOR.

Параметр BORDER управляет изображением рамки вокруг каждой ячейки и вокруг всей таблицы. При отсутствии параметра BORDER рамки не прорисовываются. При отсутствии численного значения параметра BORDER обычно оно принимается равным минимальному значению (= 1). Если BORDER = 0, ячейки располагаются как можно ближе друг к другу.

Параметр CELLSPACING определяет расстояние между рамками смежных ячеек как по горизонтали, так и по вертикали.

Параметр CELLPADDING определяет размер свободного пространства (отступа) между рамкой ячейки и данными внутри ячейки.

Параметр ALIGN определяет горизонтальное расположение таблицы в области просмотра. Допустимые значения – LEFT и RIGHT.

Пример:

```
<P><TABLE BORDER CELLPADDING=2>
<TR>
<TH>№ п/п</TH><TH>Поле 1</TH><TH>Поле 2</TH>
</TR>
<TR>
<TD ALIGN=CENTER>1</TD><TD>Ячейка 11</TD><TD>Ячейка 12</TD>
</TR>
<TR>
<TD ALIGN=CENTER>2</TD><TD>Ячейка 21</TD><TD>Ячейка 22</TD>
</TR>
</TABLE>
```

Результат интерпретации данного кода приведен на рис. 4.4.

№ п/п	Поле 1	Поле 2
1	Ячейка 11	Ячейка 12
2	Ячейка 21	Ячейка 22

Рис. 4.4. Простая таблица

Для сложных таблиц характерна потребность в объединении нескольких смежных ячеек по горизонтали или по вертикали в одну. Даная возможность реализуется с помощью параметров COLSPAN (COLumn SPANning) и ROWSPAN (ROW SPANning), задаваемых в кодах <TD> и <TH>. Форма записи: COLSPAN=num, где num – числовое значение, определяющее, на сколько столбцов следует расширить текущую ячейку по горизонтали. Аналогично ROWSPAN указывает количество строк, которые должна захватить текущая ячейка по вертикали. Пример:

```
<P><TABLE BORDER CELLPADDING=2>
<TR>
  <TH ROWSPAN=2>Объединение двух строк</TH>
  <TH COLSPAN=2>Объединение двух столбцов</TH>
</TR>
<TR>
  <TD>Ячейка 11</TD>
  <TD>Ячейка 12</TD>
</TR>
<TR>
  <TD>1</TD>
  <TD>Ячейка 21</TD>
  <TD>Ячейка 22</TD>
</TR>
</TABLE>
```

Результат интерпретации данного кода приведен на рис. 4.5.

Объединение двух строк	Объединение двух столбцов	
	Ячейка 11	Ячейка 12
1	Ячейка 21	Ячейка 22

Рис. 4.5. Сложная таблица

Создание и изменение HTML-страниц может выполняться с помощью различных средств. Существует большое количество редакторов HTML-документов. Наиболее популярными являются HotDog Web Editor (Sausage Software), Netscape Composer, Microsoft FrontPage. Два последних позволяют изменять внешний вид и компоновку страницы визуально, не вдаваясь в под-

робности реализации в виде тэгов. При этом результирующий HTML-код создается автоматически. Текстовый процессор Microsoft Word также позволяет создавать и редактировать HTML-документы.

4.6. Понятие конвейеров ввода и вывода графической информации

Множество графических приложений следуют довольно стандартной схеме построения трехмерных изображений (далее мы будем называть этот процесс конвейером). Причем некоторые программы реализуют все стадии этого конвейера, некоторые же перекладывают часть работы на плечи аппаратных устройств, специальных библиотек программ (API), другие программы или даже пользователя. Итак, конвейер состоит из следующих стадий (см. рис. 4.6).

1. *Определение состояния объектов* (Situation modeling) – эта часть программы не имеет прямого отношения к компьютерной графике, она моделирует тот мир, который будет отображаться в дальнейшем.

2. *Определение соответствующих текущему состоянию геометрических моделей* (Geometry generation) – эта часть конвейера создает геометрическое представление текущего момента нашего маленького «виртуального мира».

3. *Разбиение геометрических моделей на примитивы* (Tesselation) – эта первая действительно зависимая от «железа» стадия. На ней создается внешний вид объектов в виде набора определенных примитивов, разумеется, на основе информации из предыдущего шага конвейера.

Наиболее распространенным примитивом в наше время является треугольник, и большинство современных программ и ускорителей работают именно с треугольниками. Не будем вдаваться в математические подробности, но на треугольники всегда можно разбить любой плоский многоугольник и именно тремя точками можно однозначно задать плоскость в пространстве.

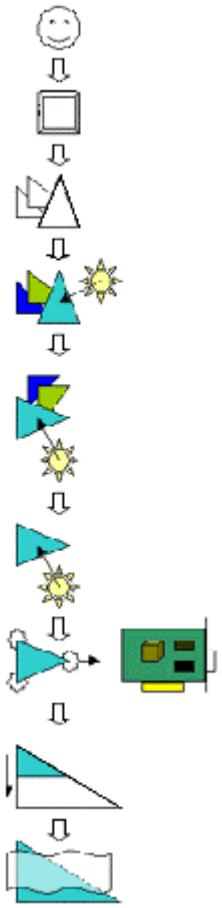


Рис. 4.6. Конвейер 3D-визуализации

4. *Привязка текстур и освещения* (Texture and light definition) – на этой стадии определяется, как будут освещены геометрические примитивы (треугольники), а также какие и как на них в дальнейшем будут наложены текстуры. Под текстурами понимаются изображения, передающие внешний вид материала объекта, т.е. негеометрическую визуальную информацию. Хороший пример текстуры – песок на абсолютно ровном пляже. Как правило, на этой стадии информация вычисляется только для вершин примитива.

5. *Видовые геометрические преобразования* (Projection) – здесь определяются новые координаты для всех вершин примитивов, исходя из положения на-

блюдателя и направления его взгляда. Сцена как бы проектируется на поверхность монитора, превращаясь в двухмерную, хотя информация о расстоянии от наблюдателя до вершин сохраняется для последующей обработки.

6**.** *Отбрасывание невидимых примитивов* (Culling) – на этой стадии из списка примитивов исключаются полностью невидимые (оставшиеся позади или сбоку от зоны видимости).

7. *Установка примитивов* (Setup) – здесь информация о примитивах (координаты вершин, наложение текстур, освещение и т.д.) преобразуется в вид пригодный для последующей стадии. (Например: координаты точек буфера экрана или текстур в целые числа фиксированного размера, с которыми работает аппаратура.)

8**.** *Закраска примитивов* **(**Fill**)** – на этой стадии собственно и происходит построение в буфере кадра (памяти, отведенной под результирующие изображение) картинки, на основе информации о примитивах, сформированной предыдущей стадией конвейера, и прочих данных, таких, как текстуры, таблицы тумана и прозрачности и пр. Как правило, на этой стадии для каждой точки закрашиваемого примитива определяется ее видимость, например с помощью буфера глубин (Z-буфера), и, если она не заслонена более близкой к наблюдателю точкой (другого примитива), вычисляется ее цвет. Цвет определяется на основе информации об освещении и наложении текстур, определенной ранее для вершин этого примитива. Большинство характеристик ускорителя, которые можно почерпнуть из его описания, относятся именно к этой стадии, так как в основном именно эту стадию конвейера ускоряют аппаратно (в случае недорогих и доступных плат).

9. *Финальная обработка* (Post processing) – обработка всей результирующей картинки как единого целого какими-либо двумерными эффектами.

Теперь попробуем разобраться, как все обстоит на самом деле. Во-первых, некоторые стадии этого конвейера могут быть переставлены местами, разбиты на части или совмещены. Во-вторых, они могут отсутствовать вообще (редко) или могут появиться новые (часто). И, в-третьих, результат работы каждой из них может быть послан (в обход других стадий) обратно. Например, картинку, полученную на последней стадии, можно использовать как новую текстуру для 8-ой стадии, реализуя таким образом эффект отражающих поверхностей (зеркал), таких как мраморный пол в игре Unreal.

4.7. Стандарты в области разработки графических систем

Программируемый интерфейс приложений (API, Application Programming Interface) состоит из функций, управляющих 3D конвейером на программном уровне, но при этом может использовать преимущества аппаратной реализации 3D в случае наличия этой возможности. Если имеется аппаратный ускоритель, API использует его преимущества, если нет, то API работает с оптимальными

настройками, рассчитанными на самые обычные системы. Таким образом, благодаря применению API, любое количество программных средств может поддерживаться любым количеством аппаратных 3D ускорителей.

Для 3D приложений существуют следующие API:

- Microsoft Direct3D;
- Criterion Renderware;
- Argonaut BRender;
- Intel 3DR.

Компания Apple продвигает свой собственный интерфейс Rave, созданный на основе их собственного API Quickdraw 3D.

Для профессиональных приложений, работающих под управлением Windows NT, доминирует интерфейс OpenGL. Компания Autodesk, крупнейший производитель инженерных САПР, разработала свой собственный API, называемый Heidi. Свои API разработали и такие компании, как Intergraph – RenderGL и 3DFX – GLide.

Существование и доступность 3D интерфейсов, поддерживающих множество графических подсистем и приложений, увеличивает потребность в аппаратных ускорителях трехмерной графики, работающих в режиме реального времени. Развлекательные приложения – главный потребитель и заказчик таких ускорителей, но не стоит забывать и о профессиональных приложениях для обработки 3D графики, работающих под управлением Windows NT, многие из которых переносятся с высокопроизводительных рабочих станций типа Silicon Graphics на PC платформу. Интернет приложения сильно выиграют от невероятной маневренности, интуитивности и гибкости, которые обеспечивает применение трехмерного графического интерфейса. Взаимодействие в World Wide Web будет гораздо проще и удобнее, если будет происходить в трехмерном пространстве.

Для производства готового результата надо определиться с двумя вещами: какая программа какие стадии конвейера будет выполнять и как она это будет делать. У нас есть три основных кандидата на работу – сама программа (как правило, начальные стадии конвейера), библиотека прикладного программмирования (интерфейс, API) и сам ускоритель. Впрочем, программы, не использующие ускоритель, все стадии конвейера выполняют самостоятельно. В понятие библиотеки в данном контексте можно (без особого зазрения совести) поместить драйвера данного ускорителя, т.к. с точки зрения программы они становятся частью библиотеки. Программ и ускорителей существует великое множество, а вот число общепризнанных библиотек весьма ограничено. Наиболее часто игры используют следующие библиотеки (см. рис. 4.7):

– *OpenGL* – созданная первоначально для профессиональных графических станций и программ трехмерного моделирования библиотека. Постепенно она пришла на платформу PC, в основном благодаря стремительному прогрессу в

области аппаратного обеспечения. Наличие поддержки этой библиотеки у ускорителя крайне желательно из-за большого числа программ, ориентированных на нее. Библиотека является в некотором роде высокоуровневой, так как берет на себя все действия, начиная с середины 4-ой ступени нашего конвейера. С одной стороны, это здорово облегчает работу программистам, с другой – способно несколько осложнить ее, особенно при реализации нестандартных эффектов или необходимости использовать новые возможности ускорителя, выходящие за рамки OpenGL. Хороший пример полезности подобного подхода – возможность выпустить версию OpenGL, значительно ускоряющую (конкретно – геометрические преобразования) работу программ на новых процессорах с SIMD наборами команд – AMD 3Dnow! и Intel SSE (Katmai New Instructions, MMX2). Очевидным достоинством также является переносимость программ на другие, не Wintel-платформы. Существенным, но быстро исправляемым недостатком – отсутствие ее полного варианта для некоторых распространенных ускорителей.

– *Direct3D* – библиотека, являющаяся частью Microsoft DirectX и поддерживаемая сейчас практически всеми ускорителями. Фактически представляет собой две библиотеки – низкоуровневую (начиная с 7-й стадии) и высокоуровневую (с 5-ой). Результат – большая гибкость для программиста в реализации его идей и, как следствие, головная боль для него же из-за множества связанных с конкретной реализацией ускорителя параметров. Сейчас идет бурное развитие этого продукта Microsoft, и, судя по всему, версия 6 Direct3D (которая уже официально вышла) будет вполне достойным конкурентом OpenGL по своим возможностям и скорости.

– *Glide* – собственная низкоуровневая библиотека (стадия 7 конвейера и далее) фирмы 3Dfx, добившаяся популярности благодаря большому распространению первых, действительно успешных ускорителей (на базе набора чипов Voodoo). Но, скорее всего, эта библиотека уйдет со сцены в ближайшие несколько лет. Она не поддерживается другими ускорителями и не будет ими поддерживаться (без разрешения 3Dfx это является незаконным).

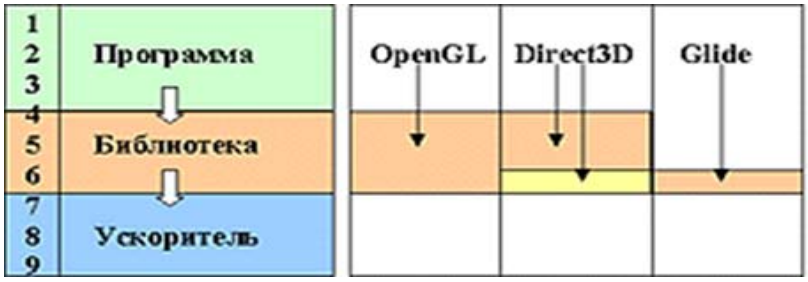


Рис. 4.7. Основные 3D библиотеки

Существует около 10 библиотек низкого уровня, созданных разработчиками различных ускорителей, такие как R-Redline фирмы Rendition, S3D Toolkit фирмы S3 и т.д. Как и следовало ожидать, программы, написанные специально для них, постепенно исчезают.

Хотя большинство современных ускорителей реализуют лишь две-три последние стадии конвейера, существует важное и быстро прогрессирующее исключение из этого правила: появились первые доступные чипы с поддержкой геометрических преобразований, способные увеличить скорость построения изображения на компьютерах с недостаточной вычислительной мощностью.

4.8. Графические процессоры, аппаратная реализация графических функций

Рынок графических подсистем до появления понятия мультимедиа был относительно прост в развитии. Важной вехой в развитии был стандарт VGA (Video graphics Array), разработанный компанией IBM в 1987 году, благодаря чему производители видеоадаптеров получили возможность использовать более высокое разрешение (640×480) и большую глубину представления цвета на мониторе компьютера. С ростом популярности ОС Windows появилась острая потребность в аппаратных ускорителях двумерной графики, чтобы разгрузить центральный процессор системы, вынужденный обрабатывать дополнительные события. Отвлечение CPU на обработку графики существенно влияет на общую производительность GUI (Graphical User Interface) – графического интерфейса пользователя, а так как ОС Windows и приложениям для нее требуется как можно больше ресурсов центрального процессора, обработка графики осуществлялась с более низким приоритетом, т.е. делалась очень медленно. Производители добавили в свои продукты функции обработки двумерной графики, такие как прорисовка окон при открытии и свертывании, аппаратный курсор, постоянно видимый при перемещении указателя, закраска областей на экране при высокой частоте регенерации изображения. Итак, появился процессор, обеспечивающий ускорение VGA (Accelerated VGA – AVGA), также известный как Windows или GUI ускоритель, который стал обязательным элементом в современных компьютерах.

Внедрение мультимедиа создало новые проблемы, вызванные добавлением таких компонентов, как звук и цифровое видео к набору двумерных графических функций. Сегодня легко заметить, что многие продукты AVGA поддерживают на аппаратном уровне обработку цифрового видео. Следовательно, если на вашем мониторе видео проигрывается в окне размером с почтовую марку – пора установить в вашей машине мультимедиа ускоритель. Мультимедиа ускоритель (multimedia accelerator) обычно имеет встроенные аппаратные функции, позволяющие масштабировать видеоизображение по осям x и y, а также аппаратно преобразовывать цифровой сигнал в аналоговый для вывода его на монитор в формате RGB. Некоторые мультимедиа акселераторы могут также иметь встроенные возможности декомпрессии цифрового видео.

4.9. 3D-акселерация

Самый общий ускоритель состоит из геометрического процессора (Geometry Processor, пока практически всегда отсутствует), механизма установки (Setup engine, стадия 7 конвейера) и механизма отрисовки примитивов – закраски (Fill engine, стадии 8 и 9), который при детальном рассмотрении оказывается комбинацией двух блоков – обработки текстур (Texel engine) и обработки буфера кадра (Pixel engine) (см. рис. 4.8).

Производительность ускорителя зависит от процессора, производительности памяти, шины и самих обрабатывающих блоков. Как правило, приводятся два числа – максимальная пропускная способность (треугольников в секунду, triangle throughput) и максимальная производительность закраски (точек в секунду, fill rate). Такой подход возможен, но не очень корректен. Да и все мы знаем, что лучшим тестом является скорость игры, в которую мы любим играть (fps, кадров в секунду на каком-то стандартном наборе действий), и качество изображения (в цифрах не измерить). Именно эти параметры и необходимо узнать в первую, но не стоит при этом забывать о сильной зависимости числа кадров от объема памяти и мощности процессора компьютера этого самого соседа.

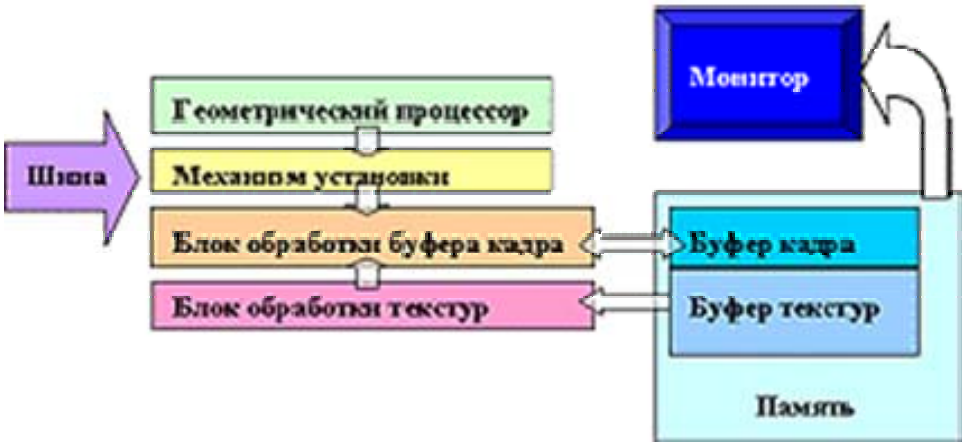


Рис. 4.8. Структура 3D-ускорителя

В общем случае закраска происходит следующим образом: блок обработки буфера кадра определяет, видна ли закрашиваемая точка, например с помощью буфера глубин (Z Buffer). Если она видна, блок обработки текстур вычисляет цвет текстуры, соответствующий этой точке примитива. Здесь есть несколько важных моментов – интерполяция (filtering, сглаживание) и выбор уровня текстуры (mip-mapping). Первый обеспечивает изображение без резких прямоугольных пикселей, даже когда вы находитесь вблизи предмета и разрешение текстуры явно недостаточно. Второй устраняет искажения (странные узоры), возникающие при чрезмерном удалении от текстуры, выбирая аналог текстуры с меньшим разрешением. Фактически, существует несколько режимов комбинирования этих двух методов. Сейчас наиболее распространены билинейная

фильтрация (bilinear filtering – вначале определяется необходимая для этого расстояния текстура, а затем – значение цвета линейно интерполируется между четырьмя соседними точками текстуры, по каждой из координат на поверхности выбранной текстуры) и три-линейная (trilinear – две билинейных для двух текстур с меньшим и большим разрешениями, затем результат интерполируется между ними). Последняя выглядит более приятно, но и требует больших затрат, так как не реализуется за один такт на современных ускорителях с одним блоком обработки текстур и, в результате, вдвое понижает скорость закраски. Сейчас появились первые ускорители с анизотропной (anisotropic – внутри куба, из двух наборов по 4 соседних точки текстур, и того по 8 точкам) фильтрацией, которая выглядит совсем хорошо, но и занимает еще вдвое больше времени. Затем вычисленный с помощью одного из вышеописанных методов цвет текстуры помещается в буфер кадра, заменяя находившееся там ранее значение, либо комбинируется с ним по какому-либо правилу (combination, blending, alpha-blending), как минимум, ускоритель должен поддерживать режимы Источник*Приемник (Src*Dest) и Источник+Приемник (Src+Dest). Подобные возможности позволяют реализовать цветное освещение, эффекты типа металла или отражения, реализовывать трилинейную фильтрацию там, где она не поддерживается аппаратно, и т.д. (Так называемое многопроходное построение изображения, multipass rendering.)

Как правило, для каждой точки текстуры, кроме RGB цвета, можно задать степень ее прозрачности (alpha, RGBA формат текстуры), которая будет использоваться ускорителем для регулирования воздействия источника на приемник.

Важны также точность представления цветов, 16 бит – Hi-Color или 32 бита – True-Color (последний гораздо лучше), и точность буфера глубин (Z-Buffer, 16 бит хуже, 24 и 32 лучше), используемого для определения видимости отдельных точек примитива.

ГЛАВА 5. ФОРМАТЫ ХРАНЕНИЯ ИЗОБРАЖЕНИЙ

Одной из важных проблем компьютерной графики является организация хранения и передачи графической информации. «Естественное» представление растрового графического изображения в виде множества пикселов очень неэкономично. Для уменьшения объема данных, представляющих изображения, используются различные *методы сжатия данных*.

Все методы сжатия данных можно условно разделить на две большие группы:

- методы без потерь информации (т.е. позволяющие полностью восстановить исходные данные из сжатого состояния);
- с потерями части незначащей, неактуальной для человека информации.

Качество метода сжатия данных определяется, прежде всего, *коэффициентом сжатия*:

$$K_{сж} = N/N_{сж},$$

где N – объем данных до сжатия; $N_{сж}$ – после сжатия.

Другой важной характеристикой метода сжатия данных является его *быстродействие*. Есть методы, которые обеспечивают большой коэффициент сжатия, но работают очень медленно, и это может оказаться критичным при использовании таких методов для работы с изображениями в режиме реального времени.

Изображения – разновидность данных. Их можно, как и любые другие данные, представить в виде набора чисел и подвергнуть обработке при помощи какого-нибудь универсального метода сжатия данных. Но известно, что универсальные методы обеспечивают хорошее качество сжатия данных лишь «в среднем», т.е. для каких-то видов данных (например, для баз данных) коэффициент сжатия может быть высоким, а для других (например, для полноцветных изображений) – незначительным.

В то же время изображения – это данные особого рода. Они обладают шириной, высотой и цветностью. Два изображения могут быть представлены двумя совершенно различными наборами чисел, но для человеческого глаза они могут выглядеть идентичными. И наоборот, два близких набора чисел могут соответствовать непохожим изображениям. В самом деле, как формально сравнить два изображения? В науке и технике широко применяются два критерия для сравнения любых абстрактных сущностей, заданных наборами $\{x_i\}$ и $\{y_i\}$ из N чисел:

- критерий среднеквадратичного отклонения $\varepsilon = \sqrt{\sum_{i=1}^{N} (x_i - y_i)^2}$;
- критерий максимального отклонения $\varepsilon = \max_N |x_i - y_i|$.

Согласно первому из этих критериев, будут мало различимы два полноцветных изображения, одно из которых идеально, а другое искажено крупными муаровыми полосами. Согласно другому из этих критериев, различными будут считаться два изображения, одно из которых идеально, а другое содержит несколько (возможно, всего даже один!) пиксел с сильно искаженной яркостью. И в том и в другом случае человек в корне не согласился бы с результатами применения формальных критериев, и был бы прав.

Все эти обстоятельства позволяют сделать вывод, что для сжатия изображений пригодны далеко не все универсальные алгоритмы сжатия данных; в то же время должны существовать методы сжатия, бесполезные для произвольных видов данных, но позволяющие эффективно сжимать именно изображения.

При изучении форматов хранения графических данных следует обращать внимание не только на их компактность, но и на удобство для дальнейшего использования. Дело в том, что большинство таких форматов являются проблемно-ориентированными, т.е. они предназначены для применения в каком-то конкретном разделе информационных технологий. Например, есть форматы, ориентированные на долговременное хранение данных на носителях; есть форматы, ориентированные на скоростной вывод изображений на экран; есть форматы, ориентированные на преобразование графической информации, находящейся в сжатом состоянии, и т.п. В процессе обсуждения мы будем отдельно останавливаться на этих вопросах.

5.1. Методы сжатия без потерь информации

5.1.1. Отсутствие сжатия. Формат BMP

В англоязычной технической литературе понятию «несжатое растровое изображение» соответствует термин «bitmap», что дословно переводится как «битовая карта». Для хранения таких изображений используется формат BMP. Этот формат разрабатывался совместно фирмами Microsoft и IBM для использования в операционных системах Windows и OS/2.

Первоначально предполагалось, что формат должен был быть универсальным и поддерживать хранение как сжатых, так и несжатых изображений. Но в конкретных реализациях (в графических редакторах и программах просмотра изображений) сжатие так никогда и не было реализовано. Общая структура BMP-файла изображена на рис. 5.1. Заголовок имеет длину 54 байта, следом за ним может располагаться цветовая *палитра* – таблица соответствия цвета его номеру. Для 256-цветных изображений она имеет длину 1024 байта, а для всех остальных типов изображений она отсутствует. Следом за палит-

Рис. 5.1. Общий формат BMP-файла

рой (или сразу за заголовком, если палитры нет) располагаются байты, описывающие изображение.

Структура заголовка изображена в табл. 5.1. Палитра, начинающаяся с байта с файловым смещением 36h, состоит из 256 элементов, каждый из которых имеет длину 4 байта. В первом байте хранится компонента B, во втором – G, в третьем – R, а четвертый байт не используется. Это очень нерационально с точки зрения компактности представления изображения, но позволяет производить быстрое считывание графической информации с диска. Минимальное значение цветовой компоненты 0, максимальное – 255, поэтому для видеорежима 320×200×256 необходимо пропорционально уменьшить все цветовые компоненты, разделив их значения на 4. Изображение хранится в файле построчно, причем порядок размещения пикселов – прямой, а порядок расположения строк изображения в файле – обратный. Это означает, что для изображения 320×200×256 в последнем байте файла хранится описание последнего пиксела первой строки картинки.

Т а б л и ц а 5.1. ***Формат заголовка BMP-файла***

Имя поля	**Смещ-е**	**Длина**	**Описание**
bfType	+00h	2	Сигнатура ‘BM’
bfSize	+02h	4	Размер файла
bfReserved	+06h	4	Не используется
bfOffBits	+0Ah	4	Позиция битов изображения
biSize	+0Eh	4	40 – размер второй части заголовка
biWidth	+12h	4	Длина строки
biHeight	+16h	4	Количество строк
biPlanes	+1Ah	2	Количество слоев
biBitCount	+1Ch	2	Количество битов на пиксел
BiCompression	+1Eh	4	0 – сжатия нет
BiSizeImage	+22h	4	Размер картинки в битах
BiXPelsPerMeter	+26h	4	Разрешение по X в пикселах на метр
BiYPelsPerMeter	+2Ah	4	Разрешение по Y в пикселах на метр
BiClrUsed	+2Eh	4	Число используемых цветов
BiClrImportant	+32h	4	0, если число цветов максимально

5.1.2. Групповое кодирование. Формат PCX

Метод группового кодирования (в англоязычной литературе *RLE – run length encoding*) использует очень простую идею: если данные представлены в виде числового потока, то каждая группа последовательно расположенных повторяющихся значений может быть закодирована парой чисел (*количество повторений*, *значение*). Коэффициент сжатия, достижимый при использовании этого метода, сильно зависит от наличия повторяющихся элементов в числовом потоке. В изображениях типа «cartoon» они встречаются очень часто, а в полноцветных изображениях могут не встретиться вообще.

Основная проблема при использовании этого метода – как закодировать сжатые данные таким образом, чтобы при обратном раскодировании отличить несжатые данные от «служебных» записей вида (количество повторений, значение). Пример решения этой проблемы демонстрируется в формате *PCX*, который широко использовался вплоть до середины 1990-х годов для хранения 2- и 4-битовых *CGA* и *EGA*-изображений (например, графических ресурсов для компьютерных игр):

- данные кодируются побайтно;
- для пометки байта-счетчика количества повторов используются 2 старших бита, они в этом случае устанавливаются в 11;
- байт несжатой информации со сброшенными старшими битами копируется в выходной поток без изменений;
- байт несжатой информации с установленными старшими битами кодируется в виде пары (*количество повторений*=1, *значение*).

Пример (см. рис. 5.2). Пусть до сжатия числовой поток состоял из 8 байтов и имел вид: 10h, C0h, C0h, C0h, C0h, C0h, 10h, C0h. Байты со значением 10h единичны, поэтому они и копируются в выходной поток без изменения. Первая группа из пяти последовательных байтов C0h может быть закодирована парой (C5h,C0h). Последний байт со значением C0h тоже единичен, но копировать его в выходной поток без изменения нельзя, так как его старшие биты равны 11 и он при раскодировании может быть перепутан со счетчиком повторений. Поэтому его приходится кодировать парой (C1h,C0h). Окончательно выходной поток состоит из 6 байтов и имеет вид: 10h, C5h, C0h, 10h, C1h, C0h. Коэффициент сжатия равен 8/6 = 1,33.

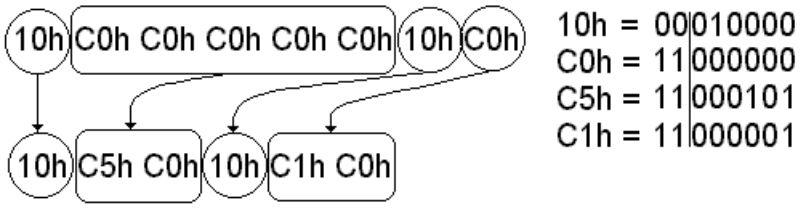


Рис. 5.2. Сжатие данных в PCX-формате

Отметим основные особенности сжатия данных, применяемые в формате PCX:

- простота алгоритмов и, как следствие, высокая скорость обработки данных;
- предельный коэффициент сжатия – 32;
- «хорошие» коэффициенты сжатия получаются для рисунков типа «cartoon», а при «неудачных» исходных данных вместо сжатия может произойти увеличение объема данных.

Разумеется, последний недостаток можно было бы частично компенсировать, просматривая изображение не только по строкам, но и по столбцам, а также по диагонали. Но это ухудшило бы быстродействие.

5.1.3. Метод сжатия LZW. Форматы GIF и TIFF

Во второй половине 1970-х годов израильские математики Абрам Лемпель (*Abraham Lempel*) и Яков Зив (*Jakob Ziv*) разработали несколько эффективных методов сжатия информации, различные модификации которых с успехом используются до настоящего времени.

Эти методы относятся к группе «словарных» методов и используют следующие обстоятельства. Сжимаемые данные очень часто содержат одинаковые фрагменты различной длины, которые, однако, могут быть расположены далеко друг от друга. Лемпель и Зив предложили собрать все такие фрагменты в одном общем «словаре», удалив их из выходного потока и заменив короткими ссылками на этот «словарь».

В середине 1980-х годов американскому математику Терри Уэлчу (*Terry Welch*) удалось усовершенствовать один из этих алгоритмов, существенно оптимизировать его по скорости и затратам памяти, а после этого запатентовать. Этот новый алгоритм, отличавшийся выдающимися характеристиками быстродействия и степени сжатия, получил наименование LZW. В начале 1990-х годов патент был перекуплен американской фирмой Unisys (производителем модемов и другой аппаратуры цифровой связи), которая активно отстаивала свои эксклюзивные права на использование этого алгоритма в течение всего срока действия патента. Это обстоятельство сильно затормозило использование алгоритма в коммерческих продуктах, но мало повлияло на применение его в продуктах класса «shareware» и «freeware». Срок действия патента истек в 2004 году.

На вход алгоритма подается поток байтов (например, байты яркостей пикселов изображения). На выходе у алгоритма образуются: 1) *словарь*; 2) *набор ссылок* на этот словарь.

Процесс сжатия заключается в следующем (см. рис. 5.3). Последовательно считываются символы входного потока, и проверяется, есть ли в созданной таблице строк такая строка. Если строка есть, то считывается следующий символ, а если строки нет, то в поток заносится код для предыдущей найденной строки, строка заносится в таблицу и поиск начинается снова.

Функция InitTable() очищает таблицу и помещает в нее все строки единичной длины (все возможные символы). Например, если сжимаются байтовые данные, то таких строк в таблице будет 256 (“0”, “1”, ... , “255”). А также добавляет в нее специальные коды. Для кода очистки (ClearCode) и кода конца информации (CodeEndOfInformation) зарезервированы значения 256 и 257 соответственно.

Функция ReadNextByte() читает символ из файла. Функция WriteCode() записывает код (не равный по размеру байту!) в выходной файл.

Функция AddStr() добавляет новую строку в таблицу, приписывая ей код. Кроме того, в данной функции происходит обработка ситуации переполнения таблицы.

В этом случае в поток записывается код предыдущей найденной строки и код очистки, после чего таблица очищается функцией InitTable().

Функция CodeStr() находит строку в таблице и выдает код этой строки.

В рассматриваемом варианте алгоритма используется 12-битный код, и, соответственно, под коды для строк остаются значения от 258 до 4095. Добавляемые строки записываются в таблицу последовательно, при этом индекс строки в таблице становится ее кодом.

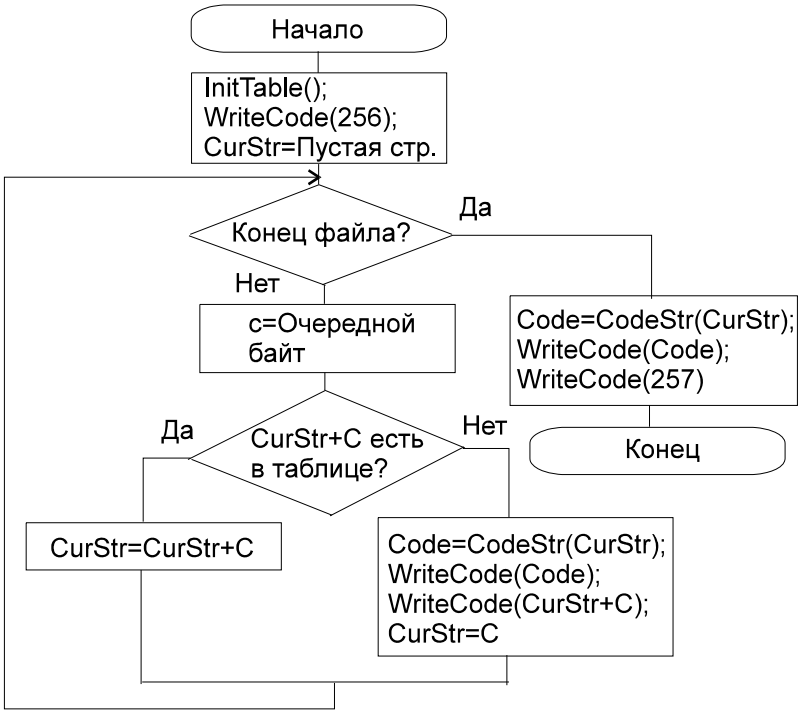


Рис. 5.3. Сжатие по методу LZW

Пример: сжать последовательность 45, 55, 55, 151, 55, 55, 55. Трассировка действий алгоритма может выглядеть следующим образом.

CurStr	**C**	**Code**
пустая строка		
45	45	
55	55	45
151	55	55
55	151	55
55, 55	55	151
55	55	
	55	259
		55

Table	
Индекс	Содержимое
0..255	0..255
256	**ClearCode**
257	**CodeEndOfInformation**
258	"45, 55"
259	"55, 55"
260	"55, 151"
261	"151, 55"
262	"55, 55, 55"

В выходной поток получим следующую последовательность: 256, 45, 55, 55, 151, 259, 55, 257.

Особенность LZW заключается в том, что для декомпрессии не надо сохранять таблицу строк в файл для распаковки. Алгоритм построен таким образом, что возможно восстановить таблицу строк, пользуясь только потоком кодов.

Легко видеть основной недостаток алгоритма LZW: он плохо сжимает короткие данные. Но по мере пополнения словаря и появления во входном потоке повторяющихся ссылок на него коэффициент сжатия быстро растет.

Метод сжатия LZW используется в форматах GIF (Graphics Interchange Format – формат обмена графикой) и TIFF (Tagged Image File Format), разработанных во второй половине 1980-х годов в компании CompuServe.

Файл формата GIF – это не просто одиночное изображение, сжатое методом LZW. Это целая «база данных», способная содержать сразу несколько изображений и набор параметров, описывающих правила их использования. Например, в заголовке GIF-файла указывается частота повторений, с которой отдельные изображения выводятся поверх другого, что позволяет создавать «мультики». В GIF-файле хранится информация о «прозрачном» цвете, который не закрывает фона при выводе изображений на экран. Перед сжатием изображения, предназначенного для хранения в GIF-формате, в нем переупорядочиваются строки, что позволяет выводить его на экран постепенно, как проявляющуюся фотографию.

Основной недостаток GIF-формата – способность хранить только 256-цветные изображения. Также GIF долгое время запрещено было использовать в коммерческих приложениях (например, в системах САПР и компьютерных играх). Тем не менее популярность GIF-формата до сих пор столь высока, что специально разработанный в середине 1990-х годов в качестве альтернативы формат PNG (PNG's Not a GIF – PNG – это не GIF) так и не смог сколько-нибудь заметно «потеснить» своего конкурента.

5.2. Методы сжатия с частичной потерей информации

5.2.1. Спектральное сжатие. Формат JPEG

Информационные сигналы в природе – результат сложения большого количества колебаний (например, звуковых или световых волн) с разными частотами и амплитудами. Разложением сигналов на суммы гармонических функций занимается спектральный анализ:

- получение суммы синусоид – преобразование Фурье;
- получение суммы прямоугольных волн – преобразование Уолша;
- получение суммы косинусоид – косинусное преобразование и т.п.

Органы чувств человека в большей степени реагируют на низкочастотные составляющие сигнала, они более информативны, чем высокочастотные. Таким

образом, практически всегда из сигнала возможно удалить часть «высокочастотной» информации, не сильно потеряв при этом в качестве сигнала. Это обстоятельство используется в графическом формате JFIF, который больше известен под общепринятым наименованием JPEG (Joint Photographic Experts Group – Объединенная группа экспертов по фотографии).

Сжатие изображений по методу JPEG выполняется в результате следующей последовательности действий.

Шаг 1. Предварительно производится перекодирование изображения из цветовой схемы RGB в схему YUV, в которой Y характеризует яркость, а U и V – красно-зеленую и сине-зеленую составляющие цвета.

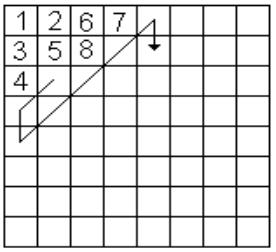
Шаг 2. Изображение разбивается на квадратные фрагменты, каждый размером 8×8 пикселов. Выборка из этих 64 чисел рассматривается как «сигнал», и к ним применяется дискретное косинусное преобразование:

$$DCT(i,j)=\frac{1}{\sqrt{2N}}B(i)B(j)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}C(x,y)\cos\left[\frac{(2x+1)i\pi}{2N}\right]\cos\left[\frac{(2y+1)j\pi}{2N}\right],$$

где $B(k)=\begin{cases}1/\sqrt{2}, k=0\\ 1, k>0\end{cases}$; $C(x, y)$ – пиксел с координатами (x, y).

В результате получается спектр. Как известно, спектральные преобразования обратимы, и никакой потери информации при этом не происходит. При восстановлении изображения используется обратное косинусное преобразование.

Шаг 3. В спектре искусственно ослабляются высокочастотные компоненты. Для этого отсчеты спектра вновь собираются в матрицу 8×8, и эта матрица почленно делится на матрицу коэффициентов ослабления. На рис. 5.4,б приведен пример такой матрицы: коэффициенты ослабления увеличиваются при движении слева направо и сверху вниз. Эта матрица не является предопределенной, ее содержимое выбирается самим пользователем в зависимости от желаемой степени сжатия (обычно при помощи «бегунка» в графическом редакторе). Чем более сильный коэффициент сжатия выбран, тем быстрее в матрице ослабления возрастают коэффициенты. Именно на этом этапе происходит потеря части информации.



3	5	7	9	11	13	15	17
5	7	9	11	13	15	17	19
7	9	11	13	15	17	19	21
9	11	13	15	17	19	21	23
11	13	15	17	19	21	23	25
13	15	17	19	21	23	25	27
15	17	19	21	23	25	27	29
17	19	21	23	25	27	29	31

а) нумерация пикселов б) матрица ослабления

Рис. 5.4. Этапы алгоритма JPEG

Шаг 5. В «ослабленном» спектре очень много повторяющихся фрагментов и нулевых значений. Поэтому матрица спектральных коэффициентов, пронумерованная по принципу «косой змейки» (см. рис. 5.4,а), вновь преобразуется в массив из 64 элементов, и к ее содержимому применяется какой-нибудь из традиционных методов сжатия (RLE, LZW и т.п., это зависит от конкретной реализации).

Объем полученных данных существенно меньше первоначального. Но чем выше коэффициент сжатия, тем выше искажения исходного изображения (см. рис. 5.5). Обычно искажения носят регулярный характер. Справедлив вывод, что применение алгоритма JPEG для сжатия штриховых изображений (отсканированных текстов, чертежей, схем и т.п.) неоправданно.

а) исходное изображение б) сжатое изображение

Рис. 5.5. Искажение изображения в результате применения метода JPEG

5.2.2. Фрактальное сжатие. Формат FIF

Фрактальная архивация основана на том, что мы представляем изображение в более компактной форме — с помощью коэффициентов системы итерируемых функций Iterated Function System (IFS).

IFS представляет собой набор трехмерных аффинных преобразований, переводящих одно изображение в другое. Преобразованию подвергаются точки в трехмерном пространстве (x_координата, y_координата, яркость).

Прежде чем рассматривать сам процесс архивации, разберем, как IFS строит изображение, т.е. *процесс декомпрессии*. Наиболее наглядно этот процесс продемонстрировал Барнсли в своей книге «Fractal Image Compression». Там введено понятие «*фотокопировальной машины»*, состоящей из экрана, на котором изображена исходная картинка, и системы линз, проецирующих изображение на другой экран (см. рис. 5.6). Машина работает по следующим правилам:

1) линзы могут проецировать часть изображения произвольной формы в любое другое место нового изображения;

2) области, в которые проецируются изображения, не пересекаются;

3) линза может менять яркость и уменьшать контрастность;

4) линза может зеркально отражать и поворачивать свой фрагмент изображения;

5) линза должна масштабировать (уменьшать) свой фрагмент изображения.

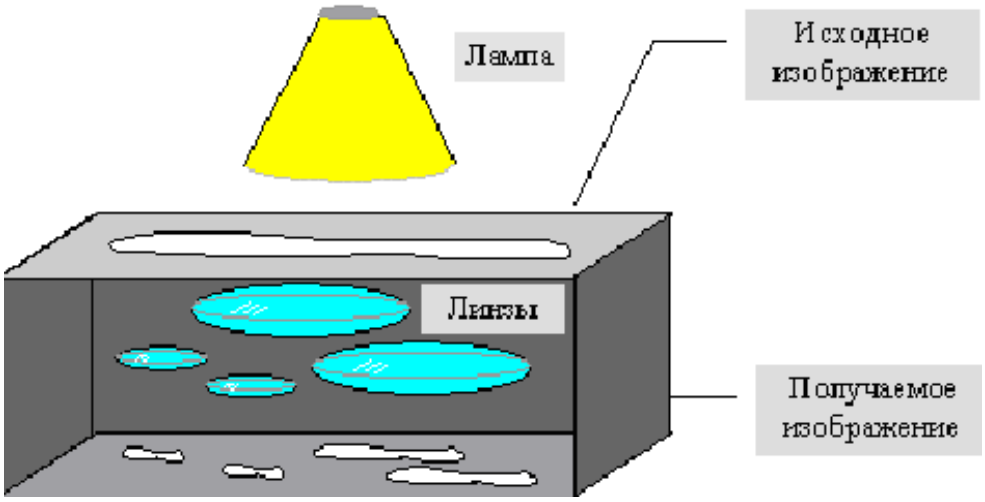


Рис. 5.6. «Фотокопировальная машина» Барнсли

Расставляя линзы и меняя их характеристики, мы можем управлять получаемым изображением. Одна итерация работы машины заключается в том, что по исходному изображению с помощью линз строится новое, после чего новое берется в качестве исходного. Утверждается, что в процессе итераций мы получим изображение, которое перестанет изменяться. Оно будет зависеть только от расположения и характеристик линз и не будет зависеть от исходной картинки. Это изображение называется «*неподвижной точкой*» или *аттрактором* данной IFS. Соответствующая теория гарантирует наличие ровно одной неподвижной точки для каждой IFS. Наиболее известны два изображения, полученные с помощью IFS, – треугольник Серпинского и папоротник Барнсли (см. рис. 5.7). Первое задается тремя, а второе – пятью аффинными преобразованиями (или, в нашей терминологии, линзами). Каждое преобразование задается буквально считанными байтами, в то время как изображение, построенное с их помощью, может занимать и несколько мегабайт.

Рис. 5.7. Фракталы «папоротник Барнсли» и «треугольник Серпинского»

Фактически, *фрактальная компрессия* – это поиск самоподобных областей в изображении и определение для них параметров аффинных преобразований. Для создания реальных алгоритмов компрессии используется ряд ограничений. Правила компрессии:

1) исходное изображение разбивается на подобласти, которые представляют из себя квадраты, которые называются *ранговыми блоками*. Ранговые блоки пересекаться не могут;

2) на исходном изображении выделяются *домены* – это совокупность 4-ранговых блоков. Домены могут пересекаться. Все ранговые блоки и домены – это квадраты со сторонами, параллельными изображению;

3) для каждого рангового блока производится попытка найти на изображении домен, такой чтобы этот домен можно было преобразовать в ранговый блок при помощи аффинных преобразований;

4) перевод домена в ранговый блок производится с помощью поворота домена на 0°, 90°, 180°, 270° и с помощью зеркального преобразования;

5) при переводе доменной области в ранговую,ее линейный размер уменьшается в 2 раза;

6) изменение яркости производится кратно некоторому коэффициенту;

7) совпадение преобразованного домена с ранговым блоком может производиться при помощи среднеквадратичного отклонения:

$$\sum (x_{дом} - x_{блк})^2 < \varepsilon_{доп},$$

где $x_{дом}$ – точка в домене; $x_{блк}$ – точка в блоке; $\varepsilon_{доп}$ – пороговое значение «похожести»;

8) если же для некоторого рангового блока не было найдено ни одного удовлетворяющего среднеквадратичному отклонению домена, то ранговый блок разбивается на 4 подобласти, и для каждой из них ищутся домены;

9) координаты, которые будут сохраняться в файл:

- координата 2 числа;
- 3 бита (номер аффинного преобразования);
- изменение яркости.

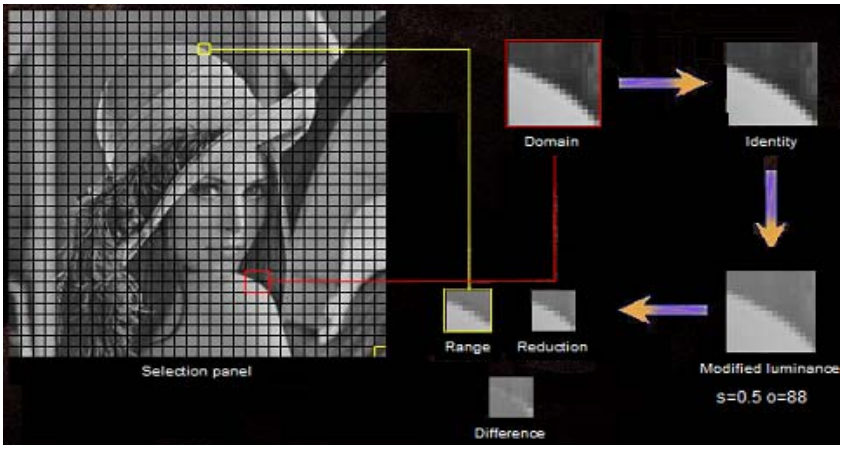


Рис. 5.8. Процесс архивации – поиск самоподобных областей

Декомпрессия алгоритма фрактального сжатия чрезвычайно проста. Необходимо провести несколько итераций трехмерных аффинных преобразований, коэффициенты которых были получены на этапе компрессии.

В качестве начального может быть взято абсолютно любое изображение (например, абсолютно черное), поскольку соответствующий математический аппарат гарантирует нам сходимость последовательности изображений, получаемых в ходе итераций IFS, к неподвижному изображению (близкому к исходному). Обычно для этого достаточно 16 итераций.

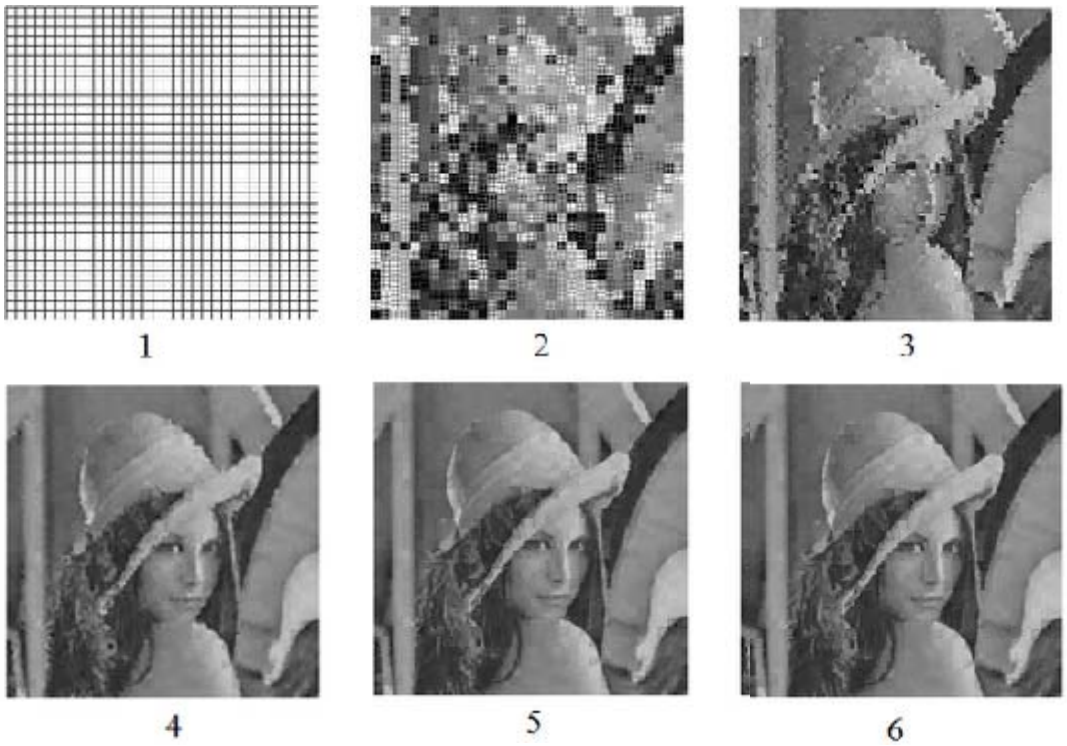


Рис. 5.9. Процесс декомпрессии изображений

На рис. 5.9. приведено начальное изображение (разлинованный квадрат) и изображения, полученные в результате первых пяти итераций работы алгоритма декомпрессии.

Формальные характеристики фрактального алгоритма следующие.

Коэффициенты компрессии: 2-2000 (задается пользователем).

Класс изображений: полноцветные 24-битные изображения или изображения в градациях серого без резких переходов цветов (фотографии). Желательно, чтобы области большей значимости (для восприятия) были более контрастными и резкими, а области меньшей значимости — неконтрастными и размытыми.

Симметричность: 100-100000. Это отношение характеристики алгоритма кодирования к аналогичной характеристике при декодировании. Характеризует ресурсоемкость процессов кодирования и декодирования. Для нас наиболее важной является симметричность по времени: отношение времени кодирования ко времени декодирования.

Характерные особенности: может свободно масштабировать изображение при разархивации, увеличивая его в 2-4 раза без появления «лестничного эффекта». При увеличении степени компрессии появляется «блочный» эффект на границах блоков в изображении.

Для обмена изображениями между исследователями в области фрактального сжатия служит официально зарегистрированный и запатентованный формат FIF.

5.2.3. Волновое сжатие. Формат JPEG2000

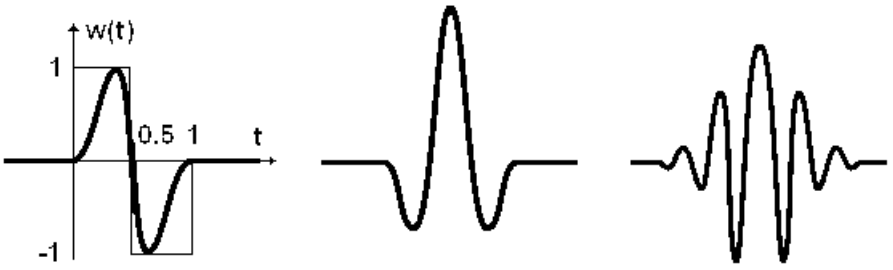
Как известно, любая пара чисел (x_1,x_2) может быть без информационных потерь заменена двумя другими числами (y_1,y_2), где $y_1=(x_1+x_2)/2$ – полусумма, а $y_2=(x_1-x_2)/2$ – полуразность исходных чисел.

Точно так же любая четверка чисел $(x_{11}, x_{12}, x_{21}, x_{22})$ может быть заменена набором $(y_{11}, y_{12}, y_{21}, y_{22})$, где

$$y_{11} = (x_{11} + x_{12} + x_{21} + x_{22}) / 4,$$
$$y_{12} = (x_{11} + x_{12} - x_{21} - x_{22}) / 4,$$
$$y_{21} = (x_{11} - x_{12} + x_{21} - x_{22}) / 4,$$
$$y_{22} = (x_{11} - x_{12} - x_{21} + x_{22}) / 4.$$

Аналогично могут быть представлены наборы из других 2^N чисел.

Коэффициенты –1 и +1 в этих формулах могут быть получены как значения так называемых вейвлет-функций (от англ. *wavelet* – всплеск) при разных аргументах. На рис. 5.10 приведены некоторые вейвлет-функции и их кусочно-постоянные аппроксимации.



а) простой вейвлет б) вейвлет МНАТ в) вейвлет Морле

Рис. 5.10. Примеры вейвлет-функций

В роли чисел ($x_{1,1}$, $x_{1,2}$, $x_{2,1}$, $x_{2,2}$) могут выступать яркости соседних пикселов, образующих квадратик 2×2. Выполнив преобразования, аналогичные вышерассмотренным, мы получим новый набор чисел ($y_{1,1}$, $y_{1,2}$, $y_{2,1}$, $y_{2,2}$), каждое из которых представляет собой результат какого-то усреднения исходных яркостей пикселов: по площади, по горизонтали, по вертикали или по диагонали. Следовательно, мы можем рассматривать каждое из этих чисел как отдельный пиксел одного из четырех новых, «усредненных» изображений (см. рис. 5.11).

Для того чтобы это можно было сделать, необходимо поступиться частью информации, содержавшейся в этом наборе, а именно: округлить получившиеся значения до ближайших целых. Разумеется, обратное преобразование теперь приведет к некоторым искажениям в исходном изображении, но оно будет небольшим. Каждое из четырех получившихся изображений можно вновь подвергнуть аналогичным преобразованиям, а потом еще и еще раз… Всякий раз будет теряться часть информации, и всякий раз в «новых» изображениях будет все сильней и сильней проявляться эффект уменьшения контрастности, т.е. перепада яркостей между соседними пикселами. Это позволяет, произведя некоторое количество итераций (обычно, 4-6), заменить пикселы разностями двух соседних. В таких наборах данных очень много нулей и повторяющихся последовательностей, что позволяет эффективно сжимать их какими-нибудь «традиционными» методами (например, RLE или LZW).

Рис. 5.11. Иллюстрация одной итерации волнового сжатия

Волновые методы обладают приемлемым быстродействием и коэффициентом сжатия. Искажения, внесенные в результате их применения, не носят регулярного характера и равномерно распределены по всей площади изображения.

Волновые методы сжатия использованы в перспективном формате JPEG2000, который, как предполагается, в течение ближайших 5-7 лет заменит собой JPEG.

5.3. Общий обзор методов сжатия графической информации

Методов сжатия изображений существует огромное количество, и далеко не все из них были нами рассмотрены. Многие существующие методы ориентированы на какой-то конкретный тип графики, например, метод JBIG идеален для сжатия монохромных штриховых изображений (текстов и схем), и поэтому применяется только при передаче графических данных по факсу, а также в формате DJVU для хранения отсканированных книг. Ряд методов основан на предварительной векторизации изображений (например, метод, использованный для хранения электронных публикаций в формате PDF). Также выпал из нашего рассмотрения ряд универсальных методов (например, метод Хаффмана, метод арифметического кодирования и пр.), которые могут использоваться для сжатия графических данных, но все же проигрывают в степени сжатия более специализированным методам.

В заключение приведем сводную таблицу, характеризующую потенциальные возможности рассмотренных методов (по данным, приведенным Д.С. Ватолиным в книге «Алгоритмы сжатия изображений»).

Т а б л и ц а 5.2. ***Характеристики методов сжатия изображений***

	RLE	**LZW**	**JPEG**	**Фракт-й**	**Волновой**
Назначение	1-, 2-, 4-бит	8- бит	8-,16-,24-бит	8-,16-,24-бит	8-,16-,24-бит
Потери	Нет	Нет	Да	Да	Да
$К_{сж}$	До 32	До 1000	2-200	2-2000	2-200

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Ватолин, Д.С.* Алгоритмы сжатия изображений [Текст] метод. пособие / Д.С. Ватолин. – М.: Изд-во ВМиК МГУ, 1999. – 76 с.

2. *Вельтмандер, П.В.* Машинная графика. [Текст]: учеб. пособие. В 3-х книгах / П.В. Вельтмандер. – Новосибирск: Изд-во НГУ, 1997.

3. *Гонсалес, Р. С.* Цифровая обработка изображений [Текст] / Р. Гонсалес, Р. Вудс – М.: Техносфера, 2006. – 1070 с.

4. Информатика [Текст]: метод. указания / сост. Ю. В. Колчин. – Самара: СГАУ, 2004. – 23 с.

5. *Климентьев К.Е.*, Методы и средства компьютерной графики [Текст] / К.Е. Климентьев, М.А. Кудрина – Самара: СНЦ-РАН, 2005. – 168 с.

6. *Краснов, М.В.* Графика в проектах Delphi [Текст]/ М.В. Краснов. – СПб: БВХ-Петербург, 2004. – 352 с.

7. *Кузан, Д.Я.* Программирование Win32 API в Delphi [Текст] / Д.Я. Кузан, В.П. Шапоров. – СПб.: БХВ-Петербург, 2005. – 368 с.

8. Методы компьютерной обработки изображений [Текст] / под. ред. В.А. Сойфера. – М.: Физматлит, 2001. – 784 с.

9. *Музыченко, В.Л.* Самоучитель компьютерной графики [Текст]: учеб. пособие / В.Л. Музыченко, О.Ю. Андреев – М.: ТЕХНОЛОДЖИ-3000, 2003 – 400 с.

10. *Петров, М.Н.* Компьютерная графика [Текст]: учеб. пособие по направлению подгот. дипломир. специалистов «Информатика и вычисл. Техника» / М. Н. Петров, В. П. Молочков. – 2-е изд. – СПб: Питер Принт, 2004. – 810 с.

11. *Порев, В.* Компьютерная графика [Текст]: учеб. пособие / В. Порев. – СПб.: БХВ-Петербург, 2002. – 428 с.

12. Программирование видеоадаптеров CGA, EGA и VGA [Электронный ресурс] / А.В. Фролов, Г.В. Фролов. – М.: Диалог-МИФИ, 1992. – 288 с.

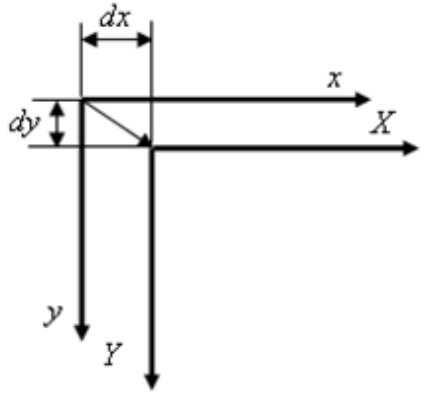
13. *Уэлстид, С.* Фракталы и вейвлеты для сжатия изображений в действии [Текст] / С. Уэлстид. – М.: Триумф, 2003. – 319 с.

14. *Фаронов, В.В.* Система программирования Delphi [Текст]/ В. Фаронов. – СПб: БХВ-Петербург, 2005. – 888 с.

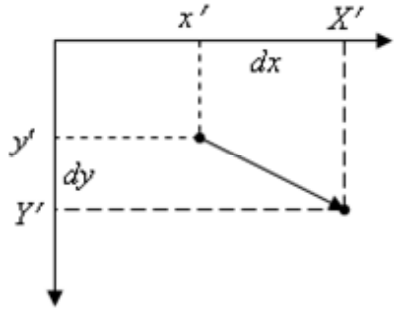
ПРИЛОЖЕНИЕ А

Основные типы аффинных преобразований систем координат и объектов на плоскости

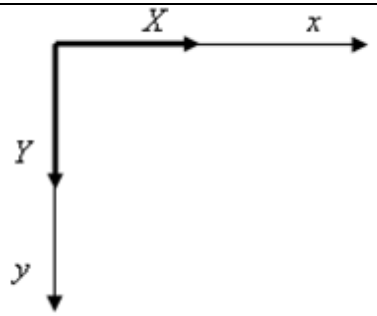
<table>
<tr><th>Преобразования системы координат</th><th>Преобразования объектов на плоскости</th></tr>
<tr><td colspan="2">Перемещение</td></tr>
<tr><td>$\begin{cases} X = x - dx, \\ Y = y - dy. \end{cases}$

$\begin{pmatrix} 1 & 0 & -dx \\ 0 & 1 & -dy \\ 0 & 0 & 1 \end{pmatrix}$</td><td> 

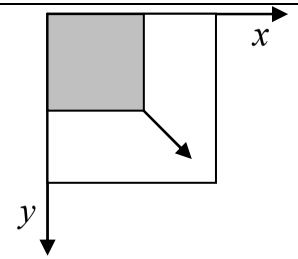
Перемещение системы координат</td><td>$\begin{cases} X' = x' + dx, \\ Y' = y' + dy. \end{cases}$

$\begin{pmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{pmatrix}$</td><td> 

Перемещение объекта на плоскости</td></tr>
<tr><td colspan="2">Масштабирование</td></tr>
<tr><td>$\begin{cases} X = x/k_x, \\ Y = y/k_y. \end{cases}$

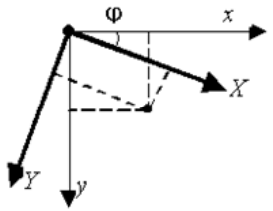
$\begin{pmatrix} 1/k_x & 0 & 0 \\ 0 & 1/k_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$</td><td> 

Масштабирование осей координат</td><td>$\begin{cases} X' = x' \cdot k_x, \\ Y' = y' \cdot k_y. \end{cases}$

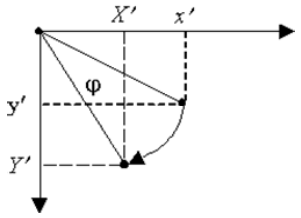
$\begin{pmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$</td><td> 

Масштабирование объектов на плоскости</td></tr>
<tr><td colspan="4">Коэффициенты k_x, k_y могут быть отрицательными. Например, $k_x = -1$ при масштабировании соответствует зеркальному отражению относительно оси y. Аналогично, $k_y = -1$ соответствует зеркальному отражению относительно оси x, а коэффициенты $k_x = k_y = -1$ будут описывать отражение относительно начала координат.</td></tr>
</table>

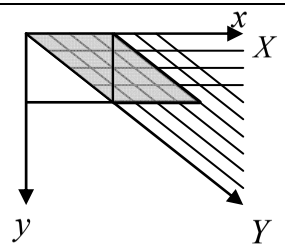
<table>
<tr><td colspan="4">Поворот
Положительный угол поворота дает поворот от оси x к оси y!</td></tr>
<tr>
<td>$\begin{cases} X = x\cos\varphi + y\sin\varphi, \\ Y = -x\sin\varphi + y\cos\varphi. \end{cases}$

$\begin{pmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}$</td>
<td> 

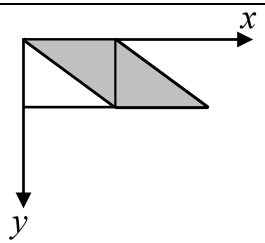
Поворот системы координат</td>
<td>$\begin{cases} X' = x'\cos\varphi - y'\sin\varphi \\ Y' = x'\sin\varphi + y'\cos\varphi \end{cases}$

$\begin{pmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}$</td>
<td> 

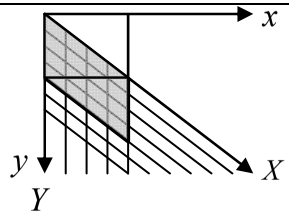
Поворот объекта на плоскости</td>
</tr>
<tr><td colspan="4">Сдвиги</td></tr>
<tr>
<td>Сдвиг вдоль оси x.
$\begin{cases} X = x - hy, \\ Y = y. \end{cases}$

$\begin{pmatrix} 1 & -h & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$</td>
<td> 

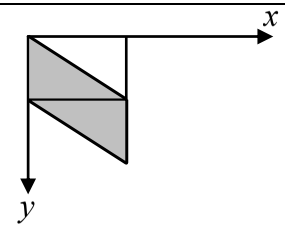
Сдвиг системы координат вдоль оси x</td>
<td>Сдвиг вдоль оси x.
$\begin{cases} X' = x' + hy', \\ Y' = y'. \end{cases}$

$\begin{pmatrix} 1 & h & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$</td>
<td> 

Сдвиг объекта вдоль оси x</td>
</tr>
<tr>
<td>Сдвиг вдоль оси y.
$\begin{cases} X = x, \\ Y = -gx + y. \end{cases}$

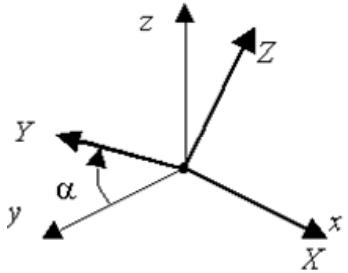
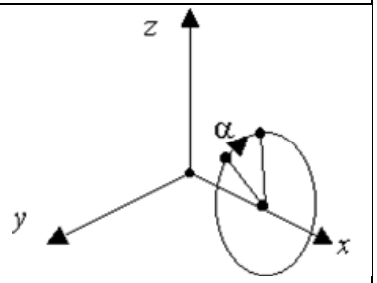
$\begin{pmatrix} 1 & 0 & 0 \\ -g & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$</td>
<td> 

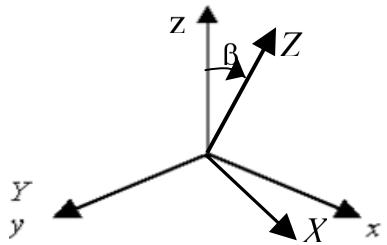
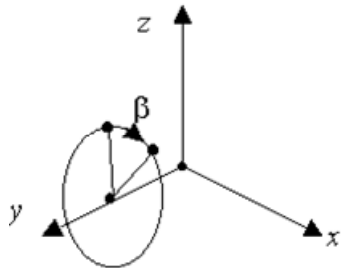
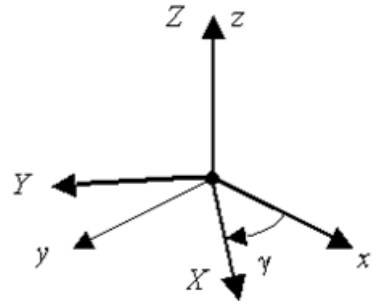
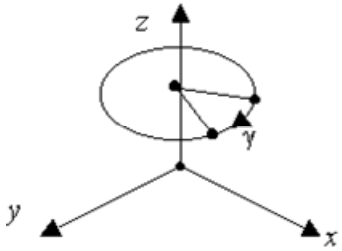
Сдвиг системы координат вдоль оси y</td>
<td>Сдвиг вдоль оси y.
$\begin{cases} X' = x', \\ Y' = gx' + y'. \end{cases}$

$\begin{pmatrix} 1 & 0 & 0 \\ g & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$</td>
<td> 

Сдвиг объекта вдоль оси y</td>
</tr>
</table>

Основные типы трехмерных аффинных преобразований систем координат и объектов

<table>
<tr><th>Трехмерные преобразования системы координат</th><th>Трехмерные преобразования объектов</th></tr>
<tr><td colspan="2">Перемещение</td></tr>
<tr><td>Перемещение системы координат
$\begin{cases} X = x - dx, \\ Y = y - dy, \\ Z = z - dz. \end{cases}$ $\begin{pmatrix} 1 & 0 & 0 & -dx \\ 0 & 1 & 0 & -dy \\ 0 & 0 & 1 & -dz \\ 0 & 0 & 0 & 1 \end{pmatrix}$</td><td>Перемещение объекта в 3D пространстве
$\begin{cases} X' = x' + dx, \\ Y' = y' + dy, \\ Z' = z' + dz. \end{cases}$ $\begin{pmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{pmatrix}$</td></tr>
<tr><td colspan="2">Масштабирование</td></tr>
<tr><td>Масштабирование осей координат
$\begin{cases} X = x/k_x, \\ Y = y/k_y, \\ Z = z/k_z. \end{cases}$ $\begin{pmatrix} 1/k_x & 0 & 0 & 0 \\ 0 & 1/k_x & 0 & 0 \\ 0 & 0 & 1/k_x & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$</td><td>Масштабирование объектов в 3D пространстве
$\begin{cases} X' = x' \cdot k_x, \\ Y' = y' \cdot k_y, \\ Z' = z' \cdot k_z. \end{cases}$ $\begin{pmatrix} k_x & 0 & 0 & 0 \\ 0 & k_y & 0 & 0 \\ 0 & 0 & k_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$</td></tr>
<tr><td colspan="2">Повороты</td></tr>
<tr><td colspan="2">При поворотах положительные значения угла вызывает вращение по часовой стрелке относительно оси, если смотреть внутрь по направлению к началу отсчета из точки, находящейся на положительном направлении оси.</td></tr>
<tr><td>а) поворот СК вокруг оси x на угол α</td><td>а) поворот объекта вокруг оси x на угол α</td></tr>
<tr><td>$\begin{cases} X = x, \\ Y = y\cos\alpha + z\sin\alpha, \\ Z = -y\sin\alpha + z\cos\alpha. \end{cases}$ $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
  </td><td>$\begin{cases} X' = x', \\ Y' = y'\cos\alpha - z'\sin\alpha, \\ Z' = y'\sin\alpha + z'\cos\alpha. \end{cases}$ $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
  </td></tr>
</table>

<table>
<tr><td>б) поворот СК вокруг оси y на угол β</td><td>б) поворот объекта вокруг оси y на угол β</td></tr>
<tr><td>$\begin{cases} X = x\cos\beta - z\sin\beta, \\ Y = y, \\ Z = x\sin\beta + z\cos\beta. \end{cases}$
$\begin{pmatrix} \cos\beta & 0 & -\sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
  </td><td>$\begin{cases} X' = x'\cos\beta + z'\sin\beta, \\ Y' = y', \\ Z' = -x'\sin\beta + z'\cos\beta. \end{cases}$
$\begin{pmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
  </td></tr>
<tr><td>в) поворот СК вокруг оси z на угол β</td><td>в) поворот объекта вокруг оси z на угол β</td></tr>
<tr><td>$\begin{cases} X = x\cos\gamma + y\sin\gamma, \\ Y = -x\sin\gamma + y\cos\gamma, \\ Z = z. \end{cases}$
$\begin{pmatrix} \cos\gamma & \sin\gamma & 0 & 0 \\ -\sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
  </td><td>$\begin{cases} X' = x'\cos\gamma - y'\sin\gamma, \\ Y' = x'\sin\gamma + y'\cos\gamma, \\ Z' = z'. \end{cases}$
$\begin{pmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
  </td></tr>
<tr><td colspan="2">Сдвиги</td></tr>
<tr><td colspan="2">Трехмерные сдвиги отличаются большим разнообразием, чем их двумерные аналоги. Матрица простейшего элементарного сдвига является единичной матрицей, в которой один ноль заменен некоторой величиной f:</td></tr>
<tr><td>Пример трехмерного сдвига:
$\begin{cases} X = x, \\ Y = -fx + y, \\ Z = z. \end{cases}$ $\begin{pmatrix} 1 & 0 & 0 & 0 \\ -f & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
В данном случае y смещено на некоторую величину, пропорциональную x, а остальные компоненты остались неизменными. Это равносильно двумерному сдвигу вдоль y.</td><td>Пример трехмерного сдвига:
$\begin{cases} X' = x', \\ Y' = fx' + y', \\ Z' = z'. \end{cases}$ $\begin{pmatrix} 1 & 0 & 0 & 0 \\ f & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$</td></tr>
</table>

Учебное издание

Кудрина Мария Александровна
Клементьев Константин Евгеньевич

КОМПЬЮТЕРНАЯ ГРАФИКА

Учебник

Редактор Ю. Н. Л и т в и н о в а
Компьютерная верстка Т. Е. П о л о в н е в а

Подписано в печать 02.04.2013. Формат 60х84 1/16
Бумага офсетная. Печать офсетная
Печ. л. 8,75.
Тираж 20 экз. Заказ № . Арт. - 3/2013.

Самарский государственный
аэрокосмический университет.
443086 Самара, Московское шоссе, 34.

Издательство Самарского государственного
аэрокосмического университета
443086, Самара, Московское шоссе, 34.

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ имени академика С. П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»**

М.А. КУДРИНА
К.Е. КЛИМЕНТЬЕВ

КОМПЬЮТЕРНАЯ ГРАФИКА

С А М А Р А 2013

ISBN 978-5-7883-0936-1