

**Виктор Алексеевич Дебелов**, доктор технических наук  
**Тагир Фаридович Валеев**, кандидат физико-математических наук

**Начальный курс компьютерной графики**  
**(черновик конспекта курса лекций)**

**Данный материал является незаконченным и невычитанным на 13.05.2011**

Новосибирск – 2008-2011

# Содержание

Содержание.....	2
Предисловие .....	5
1. Введение .....	6
1.1. Что такое МАШИННАЯ ГРАФИКА?.....	8
1.2. Основные задачи машинной графики.....	11
1.2.1. Методы графического представления машинных объектов.....	11
1.2.2. Алгоритмы преобразования машинных объектов в изображения.....	12
1.2.3. Языки описания изображений .....	12
1.2.4. Графические программные системы и библиотеки.....	13
1.3. Немного о стандартах.....	13
1.4. Выдержки из стандарта ISO/IEC 7942 (GKS) .....	13
1.4.1. Примитивы вывода и атрибуты .....	13
1.4.2. Классы логических устройств ввода .....	14
1.4.3. Виртуальная графическая станция .....	15
1.5. Графические конференции.....	16
2. Изображения, растровая графика .....	17
2.1. Дисплеи с палитрой .....	19
2.2. Полноцветные RGB дисплеи .....	20
2.3. Что такое пиксель?.....	22
2.4. Осторожно – машинная точность.....	22
2.5. Формат bmp-файла.....	23
3. Пиксельные области .....	25
3.1. Заполнение шаблоном .....	26
3.2. Аффинные преобразования над пиксельными областями.....	27
3.3. Алгоритмы заливки пиксельной области .....	28
4. Растеризация – приближение отрезков на растре .....	30
4.1. Симметричность.....	31
4.2. Наивный алгоритм .....	32
4.3. Алгоритм Брезенхэма для отрезков .....	32
4.4. Алгоритм Брезенхэма для окружности.....	34
4.5. Необходимость математической модели .....	35
5. Многоугольники .....	35
5.1. Характеристическая функция .....	36
5.2. Отсечение многоугольников.....	37
5.3. Алгоритм Сазерленда-Ходжмана .....	38
5.4. Алгоритм Вейлера-Азерттона .....	41
5.5. Регуляризованные теоретико-множественные операции .....	43
5.6. Растеризация многоугольников.....	45
5.6.1. Простейший алгоритм .....	47
5.6.2. Алгоритм ключевых ребер.....	47
6. Работа с цифровыми изображениями .....	48
6.1. Что такое пиксель?.....	48
6.2. Задача аппроксимации полутоонов.....	50
6.3. Аппроксимации полутоонов с увеличением пространственного разрешения .....	52
6.4. Аппроксимации полутоонов без увеличения пространственного разрешения .....	53
6.4.1. Попиксельное равномерное квантование .....	53
6.4.2. Попиксельный дизеринг .....	54
6.4.3. Диффузия ошибки.....	57
6.5. Гамма-коррекция.....	60
6.5.1. Гамма монитора .....	60
6.5.2. Простейшие приемы управлениями изображением .....	62

6.6. Понятие алиасинга .....	64
6.7. Алгоритм У Сяолиня .....	67
6.8. Алиасинг в анимациях .....	68
6.9. Пиксель – это квадрат? Нет – это выборка .....	69
6.9.1. Регуляризация функций, сглаживание изображений .....	70
6.9.2. Прореживание или фильтрация .....	71
6.10. Задачи обработки изображений .....	71
6.10.1. Оператор Робертса .....	72
6.10.2. Оператор Собеля .....	73
6.10.3. Выделение контуров .....	73
6.11. Замечания о цветных изображениях .....	75
6.11.1. Перевод в черно-белое .....	75
6.11.2. Дизеринг и сглаживание цветных .....	75
6.11.1. Несколько примеров, акварелизация, тиснение .....	75
6.12. Прозрачность .....	78
6.12.1. Альфа канал .....	78
6.12.2. Композиция изображений .....	78
6.12.3. Синяя комната .....	79
7. Визуализация в научных вычислениях .....	81
Wikipedia, the free encyclopedia: .....	81
7.1. Графики для презентаций .....	82
7.2. Визуализация расчетных данных .....	82
7.2.1. Одномерные графики .....	82
7.2.2. Двумерные графики .....	83
7.2.3. Функции трех переменных .....	84
7.3. Визуальное представление многопараметрических данных .....	85
7.3.1. Лица Чернова .....	85
7.3.2. Звездчатые графики .....	86
7.3.3. Кривые Эндрюса как пример функциональных представлений .....	87
7.4. Преобразование координат: вещественные $\Leftrightarrow$ растр .....	87
7.5. Изолинии, алгоритм марширующих кубиков .....	89
7.6. Векторные поля .....	93
8. Визуализация объемных плотностей .....	94
8.1. Основная гипотеза .....	96
8.1. Только поглощение .....	97
8.2. Только эмиссия .....	99
8.3. Поглощение + эмиссия .....	100
8.4. Вычисления по модели "поглощение + эмиссия" .....	100
8.5. Расчет и интерфейс .....	102
9. Элементы вычислительной геометрии на плоскости и в пространстве .....	105
9.1. Плоскость .....	105
9.1.1. Точки и векторы .....	105
9.1.2. Отрезок .....	105
9.1.3. Прямая и луч .....	106
9.1.4. Углы .....	107
9.1.5. Построение стрелки .....	107
9.2. Пространство .....	108
9.2.1. Точки, векторы, отрезки и прямые .....	108
9.2.2. Плоскость в пространстве .....	108
9.3. Симплексы и барицентрические координаты .....	109
9.4. Пересечение луча с треугольником .....	111
9.5. Разбиение единицы .....	112

9.6. Триангуляции .....	113
10. Элементы геометрического моделирования .....	115
10.1. Конструирование кривых .....	115
10.2. Параметрические кривые .....	115
10.3. Построение звена кривой .....	117
10.3.1. Сегмент кривой в форме Эрмита.....	118
10.3.2. Сегмент кривой в форме Безье .....	119
10.3.3. Самопересечения сегмента кривой в форме Безье .....	120
10.3.4. Геометрическое построение сегмента кривой в форме Безье .....	121
10.3.5. Некоторые свойства кривых Безье .....	122
10.4. Стыковка звеньев кривой .....	123
10.5. В-сплайны .....	124
10.6. Геометрическая и параметрическая непрерывность .....	125
10.7. Конструирование поверхностей .....	125
10.7.1. Параметрические поверхности .....	125
10.7.2. Поверхности Кунса .....	126
10.7.3. Задание куска в форме Эрмита .....	127
10.7.4. Задание куска в форме Безье.....	129
10.8. Несколько часто используемых форм.....	131
10.9. Некоторые необычные формы.....	133
10.10. Полигональные модели .....	136
10.11. Другие преобразования формы .....	137
10.11.1. Сведение на конус – Z-taper .....	137
10.11.2. Скручивание – Z-twist.....	137
10.11.3. Экструзия или sweeping.....	137
10.12. Представления объектов .....	138
10.12.1. Граничное представление .....	138
10.12.2. Воксельное представление .....	138
10.12.3. Конструктивная геометрия .....	138
10.12.4. Функциональное представление .....	139
10.13. Заключение .....	140
11. Преобразования координат .....	140
11.1. Введение .....	140
11.1.1. Модельные координаты и преобразования .....	141
11.1.2. Видовое преобразование .....	141
11.1.3. Анимационные преобразования .....	142
11.1.4. Сведения из векторной алгебры .....	142
11.2. Основные 2D преобразования .....	142
11.3. 2D преобразования в Postscript .....	143
11.4. Однородные координаты и 2D преобразования .....	144
11.5. Правосторонние и левосторонние системы координат.....	145
11.6. Однородные преобразования в пространстве .....	146
11.7. Перевод в другую систему координат .....	148
11.7.1. Перенос начала координат .....	148
11.7.2. Поворот системы координат .....	149
11.7.3. Преобразование мировых координат в СК камеры .....	149
11.8. Видовое преобразование .....	150
11.8.1. Параллельное проецирование.....	150
11.8.2. Перспективное проецирование.....	151
11.8.3. Преобразование видимого объема камеры к полукубу.....	152
11.8.4. Определение камеры, орты .....	154
11.8.5. Корректировка преобразования проецирования.....	155

11.9. Алгоритм Z-буфера.....	155
11.10. Вращение вокруг произвольной оси .....	156
Литература .....	157

## Предисловие

Данное пособие соответствует программе курса компьютерной графики, читаемого авторами в качестве основного курса в 6-ом семестре на кафедре компьютерных систем ФИТ и как спецкурс кафедры ФТИ ФФ. В связи с тем, что подробное изложение компьютерной графики не может быть втиснуто в программу одного учебного года, автор знакомит студентов с основными задачами и рядом базовых алгоритмов, которые используются во многих современных графических программных продуктах, и с которыми специалист, занимающийся в области информационных технологий, сталкивается почти ежедневно. Материал соответствующих лекций, конечно, шире, но информация, изложенная в данном издании, ориентирована на поддержку студентов при выполнении ими лабораторных занятий.

Материал, посвященный трехмерной и динамической графике, а также реалистической визуализации трехмерных сцен, в предлагаемый семестровый конспект лекций не вошел.

Какова цель представить именно этот материал и именно в таком объеме? В таком довольно ограниченном объеме и с ограниченной глубиной подачи материала? Ответ складывается из нескольких факторов:

- Это бакалаврский курс, всего один семестр.
- Круг разнообразных задач, относящийся к компьютерной графике, в настоящее время достаточно широк, и в один семестр его не втиснуть.
- Программа, рекомендованная министерством, вообще предлагает в одном семестре затронуть и реалистическую визуализацию. Но она ориентируется на достаточно широкий набор специальностей.
- Студенты факультета *информационных технологий*, по моему мнению, должны иметь более широкое представление о компьютерной графике по сравнению с упомянутой программой, поскольку она является одним из ключевых инструментов, например, в современных интерфейсах.
- Практика показывает, что очень небольшая часть бакалавров идет специализироваться по компьютерной графике, тогда, тем более, курс должен быть направлен на расширение кругозора, без излишних деталей, которые могут и должны быть добавлены в магистерском курсе.

Дополнительные методические материалы, включающие условия и примеры решения задач (компьютерные программы), размещены на сайте курса и могут рассматриваться в качестве приложения к пособию.

В.А. Дебелов  
Сентябрь 2009г.

## **1. Введение**

Предлагаемое пособие охватывает семестровый курс лекций по компьютерной графике. Здесь мы познакомимся с основными понятиями и задачами компьютерной графики, установим единую терминологию. За рамками этого семестра оставлены задачи реалистической визуализации пространственных сцен и задачи создания динамических приложений – анимации. По сути, предлагаемое издание – это краткий конспект лекций первого семестра курса, который читается автором в НГУ с 1998 года. Курс с каждым годом подвергался корректировке с учетом возрастающих показателей вычислительной техники (скорость, память, качество графической периферии) и программного обеспечения. В рамках курса не изучается работа с каким-либо конкретным программным продуктом типа 3D-Studio, Maya, AutoCAD, Adobe Photoshop. Все внимание уделено знакомству с наиболее часто используемыми алгоритмами, которые служат основой для разработки операций, выполняемых этими продуктами. Другими словами, вместо обучения работе на станке, мы будем изучать, как этот станок сделан.

В настоящее время трудно найти человека, который не сталкивался с тем или иным из проявлений компьютерной графики, но, тем не менее, будет полезным дать небольшой обзор истории ее развития.

Машинная (или как сейчас говорят – компьютерная) графика – это достаточно новое научное направление: почти все авторы связывают ее начало с работой И. Сазерленда 1963 года [1], где он представил первую программную разработку – редактор графических изображений Sketchpad, давший старт всему направлению. Читатель может познакомиться с этим редактором, запустив следующий ролик AlanKey\_1987\_Sketchpad\_demo.avi, который имеется среди файлов Интернет-страницы курса. Для оценки прогресса, который достигнут за несколько десятилетий, читатель может посмотреть еще один ролик multitouchreel.mpg, находящийся там же. За какие-то 5 лет проблематика машинной графики заняла определенную нишу в тематике таких конференций как весенние SJCC и осенние FJCC конференции AFIPS (American Federation of Information Processing Society). Уже в 60-х годах затрагивалась достаточно широкая проблематика. Большое место занимала тематика вывода на графические устройства – пассивная графика. С технологической точки зрения большое влияние оказала работа Брезенхэма 1965-го года [2], посвященная алгоритмам растеризации отрезков с использованием только операций сдвига и сложения. К 1973-му году накапливается значительное количество исследований и практических разработок, так, например, авиастроительные компании США разрабатывают и применяют САПР (системы автоматизированного проектирования) на комплексах IBM/360 с графическими дисплеями IBM-2250 [3].

В конце 60-х годов в нашей стране ряд научных учреждений и КБ (ИПМ АН СССР, ВЦ АН СССР, ВЦ СО АН СССР, и др.) приобретают импортные графические устройства, в основном графопостроители, а также и дисплеи – ЛВТА ОИЯИ (г. Дубна). Организации страны эксплуатировали в основном отечественную вычислительную технику, для которой еще несовершенное западное программное обеспечение было совсем неприменимо. Стали проводиться работы по инженерному сопряжению графических устройств с ЭВМ и параллельно велись разработки графических устройств (например, дисплеи ИАЭ СО АН СССР) и графического программного обеспечения.

В 1973 году выходит книга Ньюмана и Спрулла "Principles of Interactive Computer Graphics", которая переводится на русский язык и издается в СССР в 1976 году [4]. Она оказывает неоценимое влияние на развитие машинной графики в нашей стране. В это время графическая техника получает распространение в стране, особенно на машинах типа М-220 и БЭСМ-6, во многих городах ведутся разработки. На различных конференциях, связанных с

АСНИ (автоматизированные системы научных исследований) или САПР, докладываются работы по графике. В сентябре 1977 года ВЦ СО АН СССР (сейчас ИВМиМГ СО РАН) и Новосибирский филиал ИТМиВТ АН СССР проводят в Новосибирске первую конференцию, посвященную машинной графике, которая имела региональный статус, но собрала специалистов из многих районов Союза.

В 1979 году группа Siggraph ассоциации ACM (Association for Computing Machinery) анонсирует проект Core System [5], который претендует на стандарт графической системы. Этот факт показывает, что назрела насущная проблема обмена алгоритмами в форме программ, переноса (продажи) графических программных продуктов в другие организации и/или на другие компьютерные платформы. Проект оказал значительное влияние на разработки графических систем во всем мире.

В 1981-м в Новосибирске состоялась Всесоюзная конференция по проблемам машинной графики, которая имела действительно всесоюзное представительство (Москва, Владивосток, Новосибирск, Дубна, Киев, Протвино, Ленинград, Ижевск, Горький, Харьков, Пенза, Свердловск, Жуковский, Кишинев, Челябинск, Минск, Кемерово, Николаев, Ульяновск). По докладам видно влияние Core System, хотя большинство из них посвящается разработке графических алгоритмов и прикладных систем.

В 1983, наконец, официально выходит стандарт ISO Graphical Kernel System (GKS или ГКС) [6], который во многом опирается на идеи Core System, но задание самих графических примитивов отличается значительно. Многие разработчики начинают ориентироваться уже на ГКС. Забегая вперед, скажем, что с появлением IBM/PC ГКС постепенно забывается, но то огромное влияние, особенно единая терминология и большой вклад в графическое образование, нельзя не отметить. В 1988 году в СССР ГКС также становится стандартом [7].

Сильная команда из ИАиПУ ДВНЦ АН СССР, которая занималась всем спектром проблем машинной графики, проводит во Владивостоке в 1985 году очередную всесоюзную конференцию, которая даже в названии показывает расширение тематики до "распознавания изображений". В 1987 году 4-я всесоюзная конференция по машинной графике проводится в Протвино профессиональной командой по научной визуализации из ИФВЭ. А в 1989 году в Новосибирске проводится пятая и, пожалуй, последняя конференция, на которой еще серьезно рассматривались проблемы, связанные с реализацией графических систем для отечественных машин.

Очередная эра развития машинной графики в нашей стране начинается в 1991 году, когда при поддержке ACM (<http://www.acm.org/>) и Siggraph ACM(<http://www.siggraph.org/>) проводится первая международная конференция Графикон. На ней отечественные специалисты знакомятся из первых рук с наиболее передовыми разработками, в том числе: фотorealистичная визуализация, анимация, станция Silicon Graphics (<http://www.sgi.com/>). С тех пор эта конференция становится ежегодной и является крупнейшей на территории бывшего СССР ([www.graphicon.ru](http://www.graphicon.ru)).

В настоящее время в мире сложилась достаточно четкая картина ежегодных международных конференций по компьютерной графике, которые посвящены теоретическим и алгоритмическим вопросам. Отметим наиболее основные из них:

- Siggraph – самая крупная ежегодная конференция, организуемая группой Siggraph ACM (the Association for Computing Machinery's Special Interest Group on Graphics and Interactive Techniques, USA), кроме научной программы объединяет различные выставки, мультимедийные шоу, программы учебных курсов и т.п. Суммарно число участников на все мероприятия превышает 30000 человек.
- Eurographics – аналогичное ежегодное мероприятие, проводимое европейской ассоциацией компьютерной графики Eurographics.

- Графикон – аналогичное ежегодное мероприятие по компьютерной графике, проводимое на территории бывшего СССР.
- Список графических конференций можно продолжать, они проводятся также на национальном уровне в Чехии, Словакии, Польше, Турции и других странах. На многих конференциях графике посвящаются отдельные семинары.

На факультетах Computer Science западных университетов, обучающих информационным технологиям, различным дисциплинам, которые, так или иначе, связаны с компьютерной графикой, в программе отводится до 2–4 лет.

## **1.1. Что такое МАШИННАЯ ГРАФИКА?**

Машинная графика – computer graphics – в последнее время на русском обычно используется перевод термина как "компьютерная графика". Этот сначала слэнговый термин в настоящее время проник и в язык узких специалистов. Прежде всего, обратимся к словарю ISO (International Organization for Standardization) и прочтем толкование, что

**МАШИННАЯ ГРАФИКА** – это методы и средства для преобразования  
**ДАННЫХ** в (и из) **ГРАФИЧЕСКИЕ ИЗОБРАЖЕНИЯ** посредством  
**КОМПЬЮТЕРА**.

В данном определении намеренно выделены:

ДАННЫЕ,  
ГРАФИЧЕСКИЕ ИЗОБРАЖЕНИЯ,  
КОМПЬЮТЕР.

Последнее относится к слову "машинная", на языке оригинала – "computer". Мы же обратим внимание на "ДАННЫЕ" и "ГРАФИЧЕСКИЕ ИЗОБРАЖЕНИЯ". В зависимости от того, что является исходным, а что результатом, можно выделить три основных класса задач, решаемых машинной графикой, исходя из такого широкого определения, см. рис. 1.1.



Рис. 1.1. Основные классы задач компьютерной графики

**Иллюстративная машинная графика.** Этот класс задач характеризуется наличием данных, описывающих изображаемый объект. В лучшем случае это формальная модель, в худшем – некоторое эмпирическое представление, на основе которого осуществляется генерация

картины. Отметим, что описание модели отображаемого объекта намного компактнее, чем картина информаций.

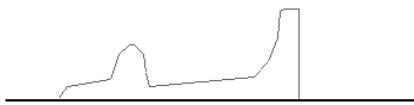


Рис. 1.2. Пример модели

На рис. 1.2 приведен пример картины, построенной по математической модели объекта, – поверхности вращения. Здесь еще придется подумать о том, какую выбрать математическую модель:  $y = f(x)$  или  $x = f_x(t), y = f_y(t)$ . А на рис. 1.3 другие графические представления на основе той же математической модели.

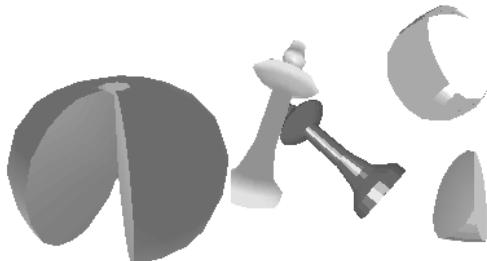


Рис. 1.3. Геометрические модели и их изображения

*Распознавание образов.* Это класс задач обратных по отношению к первому, здесь входная информация более объемна по отношению к результирующей. На основе картиныного представления решаются задачи выявления параметров некоторой предполагаемой формальной модели изображенного объекта.

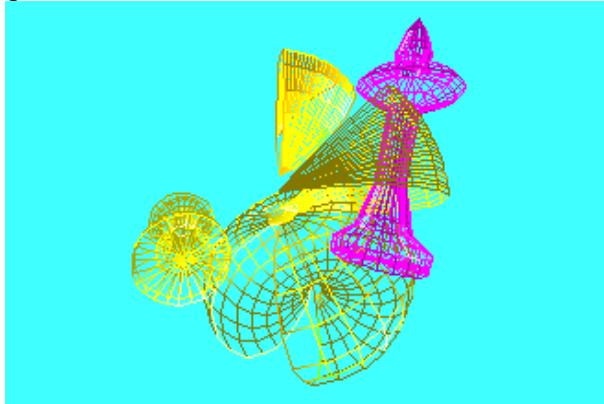


Рис. 1.4. Другой способ изображения

Очевидно, что совершенно разные алгоритмы должны разрабатываться для распознавания объектов, представленных на рис. 1.3 и рис. 1.4.

*Обработка изображений.* На входе и выходе данные одного типа – картины. При решении этих задач все равно подразумевается наличие формальных моделей изображенных объектов, что позволяет производить те или иные преобразования (обычно улучшение) картин.

Все три класса задач – все три области широкого определения машинной графики по ISO – действительно имеют много общего, например, формальные модели объектов. Разное использование этих моделей, различия в критериях при решении задач и т.п. привели к тому, что эти три направления исторически рассматриваются как самостоятельные дисциплины.

Под термином "машинаная графика" сначала понимались задачи первого типа, т.е. генерация картин по описанию, находящемуся в памяти ЭВМ. В качестве примера приведем еще одно (более узкое) определение:

*машинная графика* – это изображение *картинной информации*, генерируемой ЭВМ и *человеко-машинное взаимодействие*, которое может модифицировать это изображение.

Последнее определение можно назвать "более техническим". С другой стороны, в нем содержится достаточно информации, чтобы понять о задачах решаемых иллюстративной и *интерактивной* машинной графикой. В дальнейшем, в пределах данного курса, говоря "машинная (или компьютерная) графика", будем иметь в виду именно такой контекст.

В последнем определении мы "потеряли" понятие "формальное описание объекта". Здесь необходимо сделать следующее замечание. Формальные и/или математические модели объектов различной природы при представлении их машинными структурами данных могут совпадать. Поскольку любое формальное описание объекта, т.е. модель объекта, в ЭВМ представляется некоторыми машинными структурами данных, то мы оставляем часто за пределами нашего рассмотрения представление (моделирование) объектов структурами памяти в ЭВМ. И акцентируем внимание на графическом представлении машинных структур данных как средстве качественного анализа исходных моделируемых объектов.

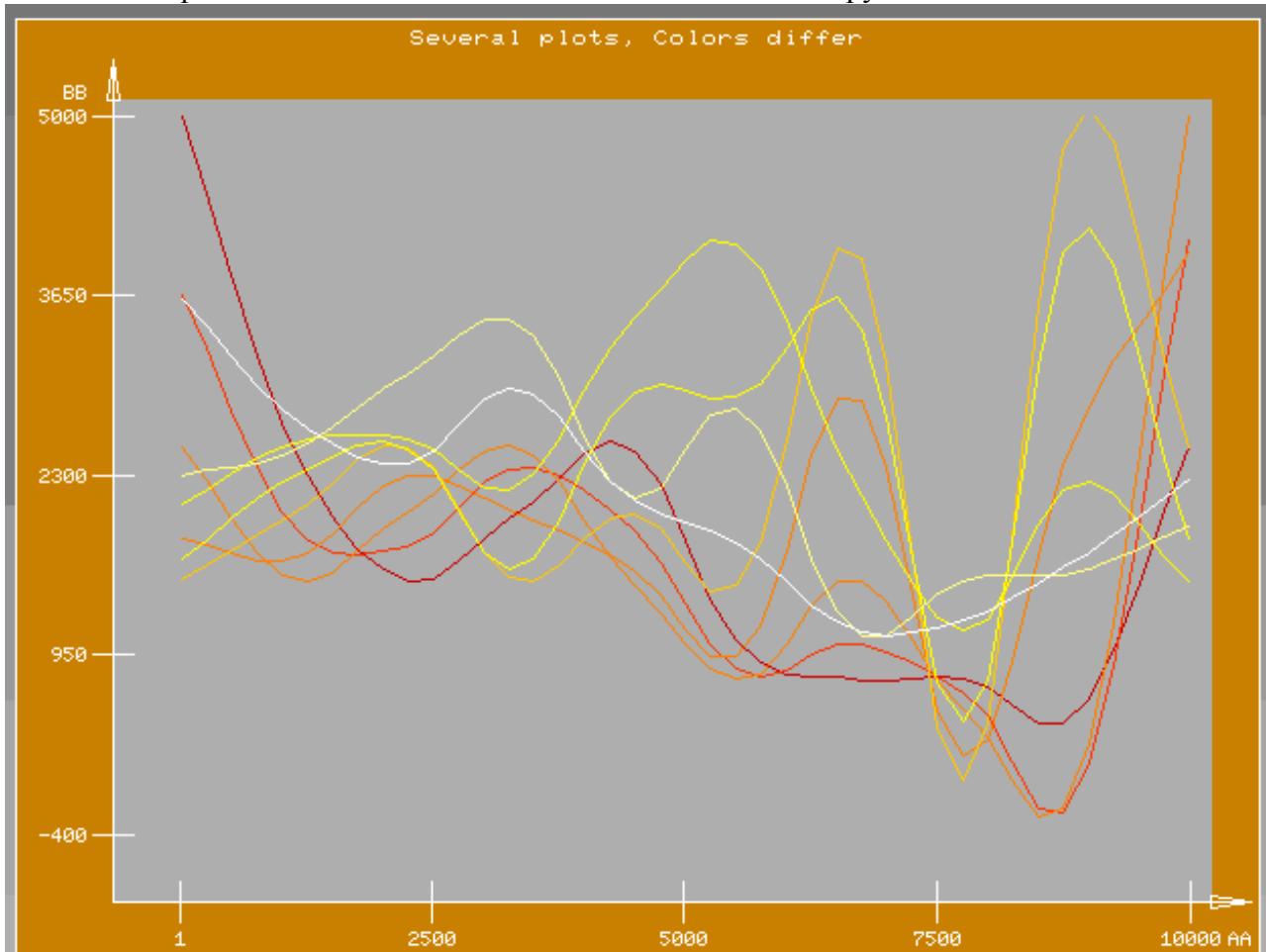


Рис. 1.5. Графическое представление 1: график в декартовой системе координат

Машинная графика получила мощную поддержку в своем развитии, благодаря:

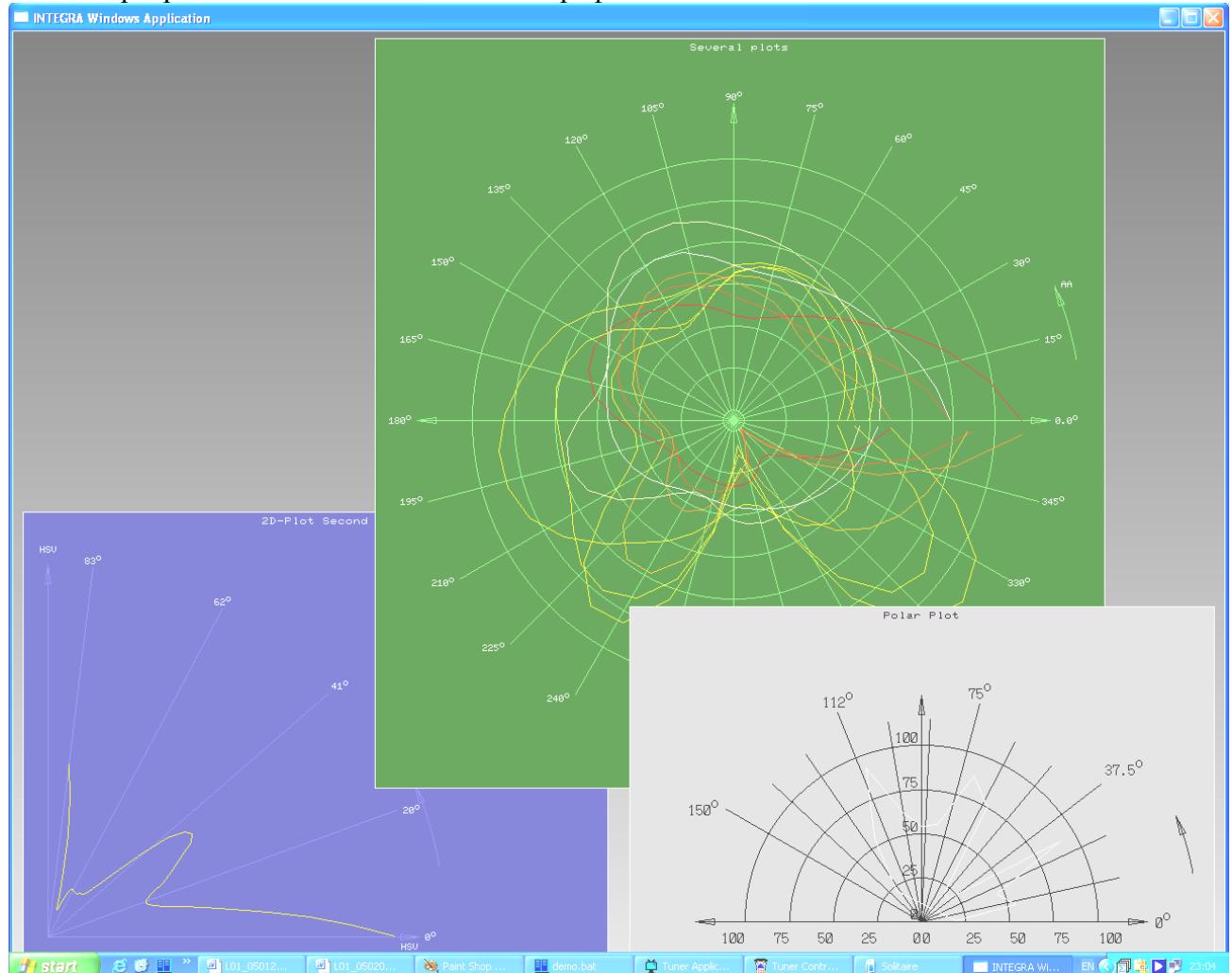
- Задачам визуализации и моделирования проектируемых и реальных объектов в Системах Автоматизированного ПРОектирования (САПР);
- Задачам визуального анализа и геометрического моделирования абстрактных математических объектов и объектов, возникающих при математическом моделировании физических процессов и явлений, т.е. исходя из нужд Систем Автоматизации Научных Исследований (АСНИ);

- Требованиям рекламы, кинематографии и компьютерных игр.

## 1.2. Основные задачи машинной графики

Автор предлагает следующее разделение задач на группы:

- Методы графического представления машинных объектов.
- Алгоритмы преобразования машинных объектов в изображения.
- Языки описания изображений.
- Программные системы машинной графики.



Таким образом, мы в своем арсенале для одномерного массива чисел имеем уже 2 представления. Разработка методов графического представления никак не связана с программированием. Это скорее заказ на программирование. Заказчик показывает вам свою модель, специфицирует ее машинное представление, рисует ее графическое представление и просит разработать алгоритм и программу, которые все это обеспечивают.

### 1.2.2. Алгоритмы преобразования машинных объектов в изображения

Это, по сути, другой этап – другие задачи. Вы уже имеете спецификации машинной структуры данных и знаете, как она должна быть нарисована, т.е. какое представление вы для нее выбираете, например, "для **double aa[100]** выбираем ГП2".

Разработчик алгоритма волен поступать так, как ему подсказывают: его квалификация; время, отпущенное ему на реализацию; частоту использования разрабатываемого алгоритма; и т.д. Следовательно, сколько независимых людей, столько, как правило, независимых алгоритмов. Конечно, только в том случае, когда разработчики хоть о чем-то заботились (время счета, необходимая память), а не делали первое, пришедшее в голову. В последнем случае алгоритмы совпадают часто.

Рассмотрим простейший пример – растеризация отрезка, т.е. дан отрезок в целочисленных координатах  $[(u_1, v_1), (u_2, v_2)]$ , необходимо его представить на растром экране. Уже сразу есть два алгоритма (мы их впоследствии рассмотрим):

1. Используя вещественную арифметику, вычислить все пиксели, заметаемые этим отрезком;
2. Алгоритм Брезенхэма.

### 1.2.3. Языки описания изображений

Под этим мы будем понимать не только, и не столько, алгоритмические языки, которые разработаны специально для графических приложений, а функциональную мощность, имеющуюся в руках разработчика алгоритма, на которую он может опираться, описывая изображения, т.е. множество примитивов и их атрибуты. Примеры:

- BitMap – точки (пиксели раstra) на дискретной плоскости;
- первые графические устройства – шаги, отрезки;
- множество инструкций дисплейного файла (добавились дуги, звезды и т.д.);
- множество форматов записей метафайлов (не один: 2 штуки в Windows, GKS, DirectX и т.д.). *Метафайл* – это файл, содержащий некоторое изображение на специальном языке;
- PostScript – язык лазерного принтера;
- Graphical Device Interface (GDI) – графические примитивы в Windows;
- OpenGL – мощная портабильная библиотека трехмерной графики;
- DirectDraw – быстрая динамическая 2D графика для создания игр;
- Direct3D (и Direct3DRM) – быстрая динамическая 3D графика для создания игр;
- Grapher – высокоуровневый пакет визуального представления расчетных данных, как правило, сеточных функций (<http://www.goldensoftware.com/>).

## **1.2.4. Графические программные системы и библиотеки**

Очень трудно проводить грань между языками и программными системами (ПСМГ), поскольку спецификация ПСМГ – это и есть его язык, а конкретная реализация – это система или библиотека. Как правило, все реализации имеют расхождения с языком, со спецификациями. И не только в сторону сужения.

Прежде, чем выбрать ту или иную ПСМГ, необходимо удостовериться, что вас устраивает ее функциональная мощность – мощность ее языка. Попробуйте необходимые функции. В противном случае вы на полпути начнете изумляться, что вы должны либо перейти на другую ПСМГ, либо достраивать уже существующую.

## **1.3. Немного о стандартах**

Большое разнообразие языков управления графическими устройствами привело к необходимости стандартизации, иначе ни о какой экономии, в смысле повторного использования, не приходится и говорить. Опираясь на один и тот же стандарт, можно готовить переносимые (портатильные) приложения, а, главное, графические алгоритмы могут стать независимыми от конкретных платформ.

Отметим, что стандарты дали единую терминологию, подвинули разработчиков аппаратуры ориентироваться на них и т.п.

Мы не будем подробно вникать в данный вопрос, отметим только, что в процессе изучения мы практически познакомимся с de facto стандартами: GDI Windows, OpenGL фирмы SGI и DirectX. По ссылке <http://www.iso.ch/iso/en/CatalogueListPage.CatalogueList> читатель может получить доступ к стандартам ISO. Далее выбрав в предлагаемом списке областей интереса "35 Information technology. Office machines", а затем "35.140 Computer graphics", читатель получит полный список графических стандартов от GKS до X3D.

## **1.4. Выдержки из стандарта ISO/IEC 7942 (GKS)**

Этот стандарт посвящен базовой графической системе – виртуальной (в общем случае интерактивной) графической станции, обеспечивающей двумерную графику. В СССР был принят соответствующий ГОСТ 27817-88 (ГКС) [7]. Охарактеризуем ГКС несколькими штрихами. Многие решения, принятые во время разработки стандарта, можно найти и в современных графических (и не только) системах и библиотеках.

### **1.4.1. Примитивы вывода и атрибуты**

*Примитив вывода (output primitive)* – базовый графический элемент, который может использоваться для построения изображения.

*Сегмент (segment)* – конечный набор примитивов.

*Изображение (display image)* – совокупность графических примитивов и/или сегментов, которая может быть одновременно выведена на носитель изображения.

*Атрибут (attribute)* – характеристика примитива вывода или сегмента, влияющая на их вид и/или на сценарий интерактивной работы. Примитивы каждого типа имеют свой набор индивидуальных атрибутов. Пользователю предоставляется возможность на их основе создать свой конечный набор виртуальных атрибутов – *связок*. Связки нумеруются и используются указанием номера. В каждой связке специфицируются конкретные значения индивидуальных атрибутов примитива. Подобное мы находим в OpenGL и GDI Windows. Специальный атрибут PickID (целое число) можно назначить каждому примитиву и сегменту. Он должен быть уникален.

Были введены следующие примитивы вывода:

1. Точечный примитив – *polymarker*. Параметры:  $\langle n \rangle, \langle x_1, y_1 \rangle, \dots, \langle x_n, y_n \rangle$  – геометрия,  $n$  точек; атрибуты – тип маркера (точка, треугольник, звездочка, ...), масштаб маркера, цвет, связка, PickID. Используется для пометки точек на изображении, для рисования точечных графиков.
2. Векторный примитив – *polyline*. Параметры:  $\langle n \rangle, \langle x_1, y_1 \rangle, \dots, \langle x_n, y_n \rangle$  – геометрия,  $n$  точек; атрибуты – тип линии, коэффициент толщины линии, цвет, связка, PickID. Представляет ломаную линию.
3. Текстовый примитив – *text*. Параметры:  $\langle x_1, y_1 \rangle$  – опорная точка текста,  $\langle string \rangle$  – строка. Атрибуты: PickID, шрифт, цвет, масштаб расширения литеры, направление текста, межлитерный просвет, точность текста, выравнивание текста, высота литеры, вертикаль литеры, связка. *Точность текста*: "до строки", "до литеры", "до штриха".
4. Полигональный примитив – *fill area*. Параметры:  $\langle n \rangle, \langle x_1, y_1 \rangle, \dots, \langle x_n, y_n \rangle$  – геометрия,  $n$  вершин. Атрибуты: PickID, связка, вид заполнения (шаблон/штриховка/цвет), размер шаблона для заливки шаблоном, точка привязки шаблона, матрица шаблона, вид штриховки, цвет, оформление бордюра. Очень похоже на то, как MS Word позволяет оформлять многоугольники.
5. Растровый примитив – *cell array*. Параметры:  $\langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle$  – прямоугольник;  $m, n$  – разбиение на клетки по горизонтали и вертикали; *color* – массив из  $m \times n$  цветов. Атрибуты: PickID.
6. Обобщенный примитив черчения – *GDP*. Атрибуты: PickID. Этот примитив служит для программирования нестандартных действий – для использования аппаратных возможностей конкретных устройств. Параметры и атрибуты зависят от конкретной реализации.

Понятие изображения в ГКС строилось на основе широко распространенного в то время *сегментированного дисплейного файла* (СДФ), представлявшего упорядоченную последовательность команд, с помощью которых формировались примитивы и сегменты, задавались координатные преобразования. Команда вызова сегмента позволяла существенно экономить память.

#### **1.4.2. Классы логических устройств ввода**

Станция ГКС оснащена возможностями ввода информации за счет наличия логических устройств ввода. Логическое устройство предоставляет прикладной программе значение логического ввода. Тип этих данных определяется классом ввода.

*Логическое устройство ввода* является обобщением одного или более физических устройств и служит для передачи в программу значений логического ввода.

*Значение логического ввода* – значение, связанное с логическим устройством и преобразованием передаваемых данных.

*Класс ввода* – логически эквивалентный по выполняемым функциям набор устройств ввода.

Рассмотрим классы ввода ГКС.

1. Ввод позиции – *locator*. Предоставляет прикладной программе координаты точки  $\langle x, y \rangle$  в мировых координатах, т.е. в системе координат приложения.

2. Ввод последовательности позиций – *stroke*. Предоставляет прикладной программе последовательность координат точек в мировых координатах:  $\langle n \rangle$ ,  $\langle x_1, y_1 \rangle$ , ...,  $\langle x_n, y_n \rangle$ .
3. Ввод числа – *valuator*. Предоставляет прикладной программе вещественное число.
4. Выбор альтернативы – *choice*. Предоставляет прикладной программе неотрицательное целое число из некоторого диапазона.
5. Указание – *pick*. Предоставляет прикладной программе PickID сегмента и PickID примитива, на которые было произведено указание оператора.
6. Ввод строки – *string*. Предоставляет прикладной программе последовательность литер.

Каждое логическое устройство ввода может функционировать в одном из трех возможных *режимов ввода*, который определяется прикладной программой. Одновременно устройство может находиться только в одном из режимов:

*Запрос* (request) – обращение к специальной функции в этом режиме вызывает попытку прочесть значение логического ввода с указанного логического устройства. ГКС ожидает ввода до тех пор, пока оператор либо не введет данные, либо не выдаст сигнал отмены ввода.

*Опрос* (sample) – обращение к специальной функции в этом режиме предписывает ГКС возвратить текущее значение логического ввода с указанного логического устройства, не дожидаясь действий оператора. Устройство должно находиться в режиме ОПРОС.

*Событие* (event). ГКС поддерживает одну входную очередь событий, состоящую из упорядоченных в порядке поступления записей о событиях. Запись о событии содержит идентификатор логического устройства и значение логического ввода, полученное от этого устройства. Записи о событиях генерируются асинхронно, исключительно в зависимости от действий оператора.

Каждое логическое устройство ввода не обязательно поддерживается каким-либо одним физическим устройством. Это программно-технический комплекс, предназначенный для ввода данных требуемого логического класса ввода. Оно управляет оператором, которому обеспечивается *эхо* и *подсказка*, возможность подать сигнал подтверждения или отмены ввода.

*Эхо* (echo) – немедленное оповещение оператора о текущих значениях, которые обрабатываются устройством ввода (курсор, перекрестье, шкала и т.д.).

*Подсказка* (prompt) – выводимая оператору информация, указывающая на доступность данного логического устройства ввода.

*Подтверждение* (acknowledgement) – вывод, оповещающий оператора, что его действие, отметившее определенный момент времени в процессе взаимодействия, принято системой во внимание.

### 1.4.3. Виртуальная графическая станция

Основным понятием стандарта ГКС является графическая станция, которая предоставляет возможности по выводу изображений и возможности по обеспечению интерактивной работы. Более того, ГКС – это мульти-станция, т.е. одна копия ГКС управляет всеми станциями на конкретной платформе. Отметим некоторые из задач, на решение которых была направлена разработка ГКС:

синтез и воспроизведение изображений;  
направление частей изображений, определенных в различных пользовательских системах координат, на различные графические станции и преобразование их координат в координаты соответствующих устройств;  
управление станциями, к которым имеет доступ система;  
обслуживание ввода данных со станций;  
поддержка разбиения изображений на части, которые можно независимо обрабатывать (рисовать, преобразовывать, копировать, удалять);  
долговременное хранение изображений.

С точки зрения мощности средств вывода различаются следующие уровни функциональности:

*Уровень вывода 0*: обеспечивается базовая графика. Графический метафайл поддерживать не обязательно.

*Уровень вывода 1*: несколько станций, полная концепция связок атрибутов, работа с метафайлом, сегментация изображений.

*Уровень вывода 2*: общая память сегментов для станций.

С точки зрения мощности системы ввода для станции определены уровни:

*Уровень ввода a*: устройства ввода отсутствуют. Пассивная станция.

*Уровень ввода b*: имеются устройства ввода, обеспечивающие ввод всех классов логического ввода. Ввод осуществляется в режиме *запрос*.

*Уровень ввода c*: имеются устройства ввода, обеспечивающие ввод всех классов логического ввода. Ввод осуществляется во всех трех режимах.

Теперь, специфицируя свою программу, пользователю достаточно было сказать, что она требует станцию ГКС уровня 2b, т.е. станция должна обеспечить полную функциональность по выводу и ввод в режиме запрос. Фактически станция ГКС – это программно-технический комплекс. Например, вполне возможно гипотетически представить реализацию станции уровня 2b, построенную на основе клавиатуры и дисплея. Прототипом станции ГКС послужила наиболее распространенная конфигурация, состоящая из сателлитной мини-ЭВМ и набором дополнительных устройств: векторные и растровые дисплеи и графопостроители, принтеры, сколки и сканеры, фотопостроители, станки с ЧПУ, фрезы, различные функциональные клавиатуры и др.

## **1.5. Графические конференции**

*The SIGGRAPH Annual Conference* (<http://www.siggraph.org/conference/>) Ежегодная конференция, которая проводится группой Siggraph ассоциации ACM. Это самая крупная конференция по компьютерной графике в мире. Кроме обязательной программы докладов по новым разработкам, результатам и достижениям в области КГ, структура конференции традиционно включает: STAR (state of art reports) – доклады, отражающие текущее состояние дел в отдельной узкой области КГ; школу – порядка 40 курсов по различным тематикам из области КГ; промышленные презентации; и т.д. Число участников превышает 30 тыс. человек.

*The EUROGRAPHICS Annual Conference* (<http://www.eg.org/>). Аналогичная конференция, проводимая под эгидой Европейской ассоциации КГ. Структура конференции в целом повторяет предыдущую. Да и состав ключевых участников ( оргкомитеты и приглашенные доклады) также сильно коррелирует.

WSCG 2010 - conference on Computer Graphics, Visualization and Computer Vision (<http://wscg.zcu.cz/>). Зимняя европейская конференция проводится в Чехии.

GraphiCon ([www.graphicon.ru](http://www.graphicon.ru)). Крупнейшая (а может и единственная) международная конференция по КГ на пространстве бывшего СССР проводится с 1991 года.

Имеется очень много (десятки) конференций по различным дисциплинам, на которых значительное внимание уделяется КГ – это либо отдельные секции, или более узкая тематика, например, графический интерфейс. Многие страны проводят свои национальные конференции по КГ: Словакия, Турция и т.д.

## 2. Изображения, растровая графика

В первую очередь встает вопрос: "Что такое изображение?". На рис. 2.1. даны два примера изображений.

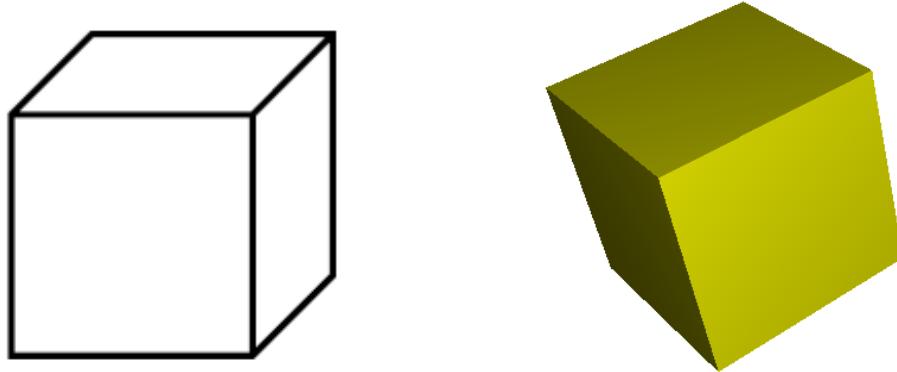


Рис. 2.1. Примеры изображений

Обе картинки представляют куб, но совершенно по-разному. Базируясь на правом рисунке на рис. 2.1 дадим следующее очень общее определение. Если не говорится что-либо иное, то в самом общем случае под изображением мы будем понимать некоторое распределение световой энергии на *плоском* носителе (фотопленка, бумага, экран монитора и т.д.), когда в каждой точке  $(x, y)$  для каждой длины волны  $\lambda$  задано значение  $E(x, y, \lambda)$ . Для динамических изображений –  $E(x, y, \lambda, t)$ , т.е. вводится зависимость от времени.  $E$  – непрерывная функция. Тем не менее, в дальнейшем мы удостоверимся в том, что это определение справедливо и для левого проволочного изображения кубика, поскольку определяется изображение, а не то, что на нем изображено.

Компьютерная графика имеет дело с изображениями, хранящимися в памяти ЭВМ, и здесь важную роль начинают играть языки описания изображений. Самые нижние уровни языка базируются на аппаратном решении воспроизведения изображений.

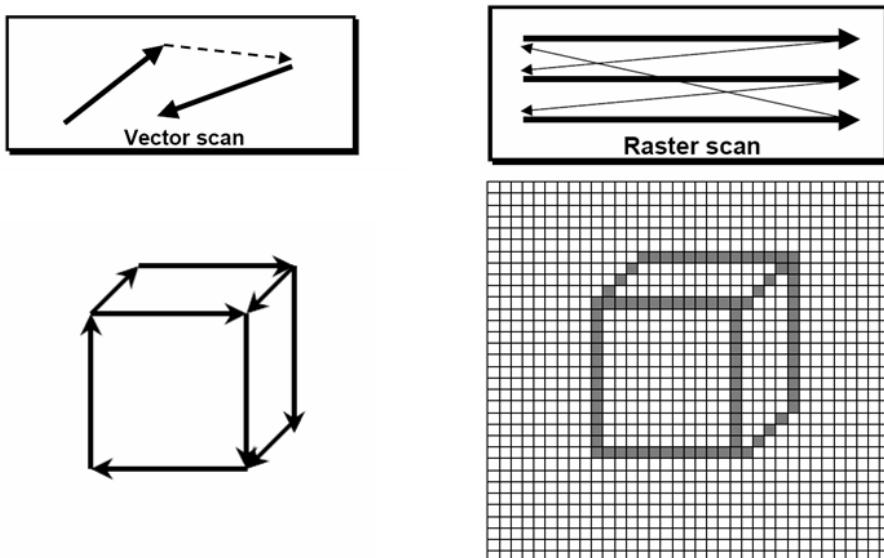


Рис. 2.2. Векторный и растровый способы отображения

Как правило, координаты точек задаются на некоторой равномерной сетке. Исторически применялись два различных способа для вывода изображений по их описаниям:

- *Векторный*, рис. 2.2, слева. Изображение строится по опорным точкам примитивов (здесь отрезки), путь луча (или пера графопостроителя) полностью повторяет инструкции дисплея. Множество инструкций, описывающих изображение, называется *дисплейным файлом*.
- *Растровый*, рис. 2.2, справа. Изображение представляет собой матрицу или *растр* из одинаковых элементов, в идеале – квадратиков, которые получили название *пикселей* (pixel – picture element). Вывод этой матрицы на устройство (экран дисплея или принтер) осуществляется построчно из памяти машины. Подобным образом устроена телевизионная развертка.

Отметим, что в обоих случаях координаты опорных точек не произвольные, а привязаны к узлам некоторой равномерной сетки на плоскости. Исторически векторные дисплеи появились раньше и лишь потом были вытеснены растровыми. Основная причина этому кроется в дороговизне оперативной памяти: для вывода несложных векторных изображений требовалось хранить значительно меньше информации. Даже для хранения небольшого растра, к примеру,  $320 \times 240$  с глубиной цвета 1 бит требуется 8 Кб ОЗУ, что на заре развития вычислительной техники порой являлось непозволительной роскошью. С другой стороны для хранения одного отрезка с координатами от 0 до 255 достаточно 4 байт, то есть довольно сложное изображение в 100 отрезков потребует всего 400 байт памяти. Однако со временем память стала более доступна и теперь растровый способ отображения используется практически везде, хотя и добавляет проблемы, которые отсутствовали у векторных дисплеев.

Таблица 2.1. Сравнение векторной и растровой графики

### Векторная графика

Размер дисплейного файла напрямую зависит от сложности изображения.

### Растровая графика

Размер растра не зависит от изображения. Запоминается каждый пиксель независимо от информативности.

Удобно для вывода чертежей. Нет проблемы алиасинга. Невозможно выводить фотографии, заливку с градиентом и т. д.

Можно зарисовывать точку дважды. После удаления одной из пересекающихся линий непрерывность рисунка остается.

В связи с тем, что растровая графика более универсальна, можно говорить, что векторная графика для представления изображений на дисплее на современном этапе "умерла", хотя это не означает, что она не используется на более высоких уровнях представления изображений. Далее мы будем рассматривать вывод только на растровые устройства.

Можно сказать, что изображению – раству на экране – соответствует матрица, элементы которой содержат характеристику цвета соответствующего пикселя раstra. Раstrу  $N \times M$  пикселей соответствует матрица  $N \times M$  значений в памяти машины, называемая *буфером кадра* (frame buffer). Изменения значений в буфере кадра "моментально" отображаются на экране (если быть точным, то при следующем обновлении, частота которых называется частотой развёртки в настройках видеорежима). Поскольку часто из контекста понятно о чём идет речь, то вместо значения элемента буфера кадра говорят о значении или цвете пикселя. Под значение характеристики цвета пикселя может отводиться 1, 2, 4, 8, 12, 15, 16, 24, 32 и более разрядов. Как правило, в первых четырех случаях для отображения на экране дисплея используется палитра, остальные случаи относятся к так называемым полноцветным дисплеям. Режим использования палитры иногда называется Ramp-mode, например, в DirectX.

Часто применительно к растровым изображениям (или просто – к *растрам*) используется термин *разрешение* (resolution). Оно косвенно характеризует размер пикселя. Одно и то же изображение может быть представлено растром  $N_1 \times M_1$  и растром  $N_2 \times M_2$ . Тогда говорят, что первый растр имеет большее разрешение, если  $N_1 > N_2$  и  $M_1 > M_2$ .

## 2.1. Дисплеи с палитрой

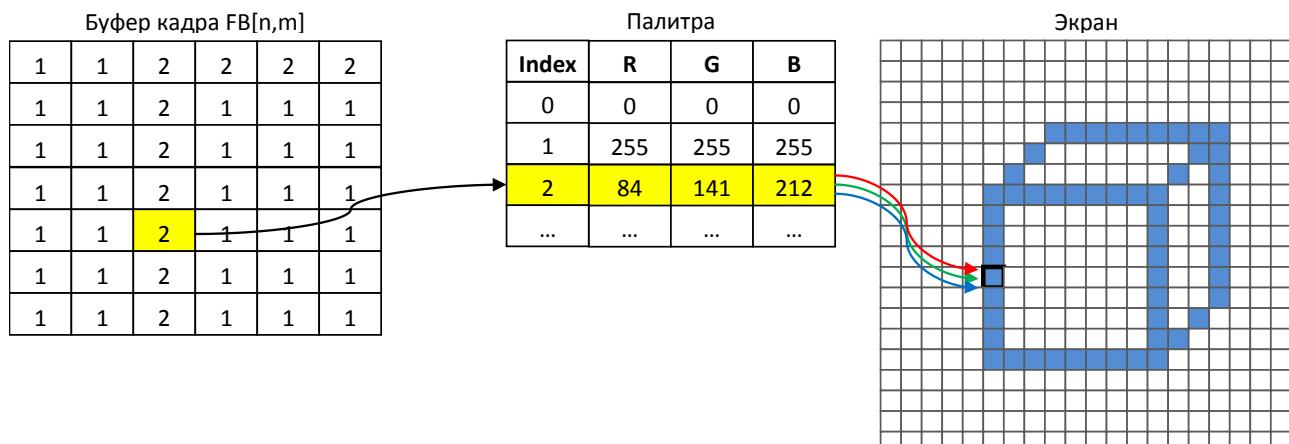


Рис. 2.3. Работа дисплея с палитрой

Если для значения цвета пикселя используется от 1-го до 8-ми битов, то максимальное число различных цветов на изображении соответственно от 2-х до 256-ти, значения: 0–1, ..., 0–255.

В аппаратуре дисплея имеется специальная память под таблицу цветности с аналогичным числом строк (2, ..., 256). Эта таблица называется *палитрой* или LUT (Lookup Table). В каждой ее строке записана характеристика цвета пикселя. В буфере кадра для каждого пикселя хранится не характеристика цвета пикселя, а номер строки палитры.

Тогда работа дисплея по отображению идет по следующему алгоритму:

- Переход к отображению очередного пикселя  $[i, j]$ .
- Выбор соответствующего значения из буфера кадра  $q = FB[i, j]$ .
- По  $q$ -й строке таблицы цветности формируется цвет на основе записанных в ней значений для красной (R), зеленой (G) и синей (B) электронных пушек.
- Сформированный луч подсвечивает пиксель  $[i, j]$ .

Наличие команд для программирования палитры позволяет на 2-х битовом дисплее воспроизводить любые цвета, но не более 4-х одновременно. Смена значений RGB какой-либо из строк палитры немедленно отражается на экране, например, все красные пиксели становятся синими.

Первые персональные компьютеры выводили растр с однобитовой глубиной цвета без возможности настройки палитры. Существенным шагом вперёд стал графический адаптер CGA, выпущенный IBM в 1981-м году. Он обладал 16 Кб видеопамяти и позволял выводить до четырёх цветов одновременно в режиме  $320 \times 200$ , причём с некоторыми ограничениями палитру можно было настраивать. При более высоком разрешении  $640 \times 200$  глубина цвета была ограничена одним битом, так как на большее не хватало видеопамяти. Заметим, что при таком разрешении пиксели оказывались прямоугольными — вытянутыми по вертикали. В 1984-м году появился адаптер EGA уже с 64 Кб видеопамяти (более поздние версии с 256 Кб), где палитра могла включать 16 цветов (4 бита). Восьмибитная глубина цвета впервые появилась на персональных компьютерах в режиме  $320 \times 200$  адаптера VGA (1987-й год). Существовали также более экзотические варианты. К примеру, компьютер Commodore Amiga поддерживал режим 6-битной палитры, а некоторые станции Silicon Graphics — 12 бит.

В связи с постоянным удешевлением памяти графические адAPTERы с палитрой канули в лету, используются в основном полноцветные режимы. Хотя, необходимо заметить, что для ряда приложений были разработаны даже специальные алгоритмы, которые существенно использовали наличие программируемой палитры.

## 2.2. Полноцветные RGB дисплеи

У полноцветных дисплеев каждое значение в буфере кадра напрямую указывает цвет, которым будет закрашен соответствующий пиксель. Так, например, если значение занимает 15 разрядов, то они распределяются по 5 разрядов на каждый канал цвета — красный, зеленый, синий. Если 24, то по 8 разрядов. На некоторых моделях сотовых телефонов, карманных плееров и КПК использовался 12-битный полноцветный режим.

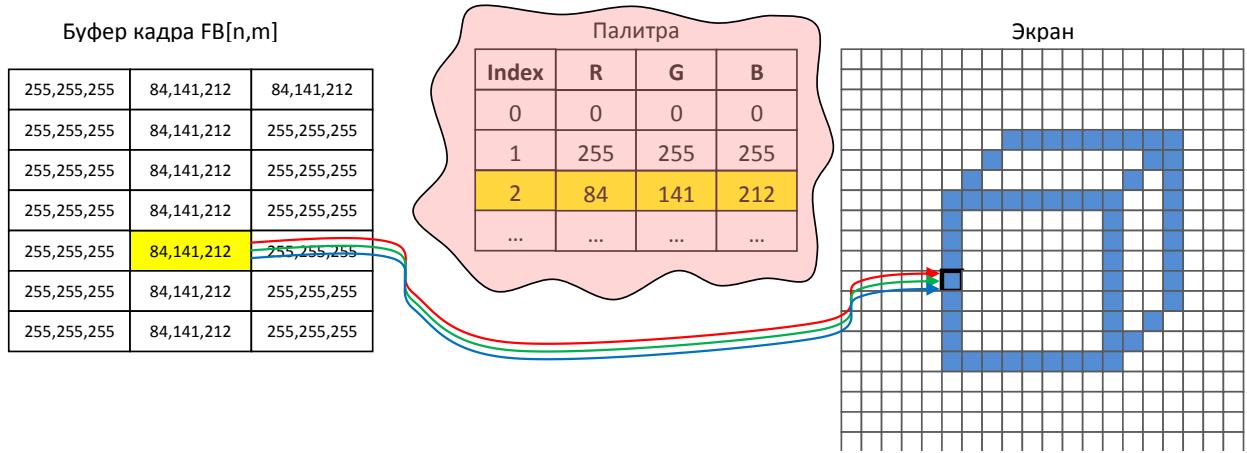


Рис. 2.4. Убирается из аппаратуры палитра, но остается в программном арсенале

Работа полноцветного дисплея подчиняется следующему алгоритму:

- Переход к отображению очередного пикселя  $[i, j]$ .
- Выбор соответствующего значения из буфера кадра  $q = FB[i, j]$ . **Не понял, что поменяно в этой строке - Деб**
- По значению  $q$  формируется цвет на основе записанных в нем значений для красной (R), зеленои (G) и синей (B) электронных пушек.
- Сформированный луч подсвечивает пиксель  $[i, j]$ .

32-х разрядное значение интерпретируется как цвет аналогично 24-х разрядному значению, плюс 8 разрядов отводится под альфа-канал (непрозрачность), о котором мы поговорим позже. Иногда последние 8 разрядов просто не используются. Однако их удобно оставлять, так как цвета выравниваются по 32-разрядным машинным словам, что позволяет эффективно их обрабатывать на низком уровне (к примеру, с использованием инструкций MMX).

Наибольшее число разрядов значения в буфере кадра было изначально аппаратно спроектировано в дисплеях фирмы Silicon Graphics Inc. ([www.sgi.com](http://www.sgi.com)) – 96, которые для каждого пикселя распределены таким образом:

32 разряда: RGB + альфа-канал;  
16 разрядов: буфер глубины или z-буфер (рассмотрим позже).

И аналогичные 48 разрядов для дублирующего буфера. Другими словами, аппаратный буфер кадра состоит из двух одинаковых буферов, один из которых – передний – отображается на экране, а второй – задний – не отображается. Такая организация буфера кадра ориентирована на поддержку *метода двойной буферизации*. Данный метод является основой для программирования динамических изображений или анимаций. Каждый момент времени в пикселе экрана отображается содержимое первых или вторых 48-ми битов. Суть двойной буферизации в следующем алгоритме:

- Очередной кадр фильма формируется в заднем буфере (back buffer).
- По завершению формирования кадра производится переключение страниц (page flipping), когда задний буфер становится передним (front buffer), т.е. отображаемым, а передний – задним, т.е. предназначенным для модификации.

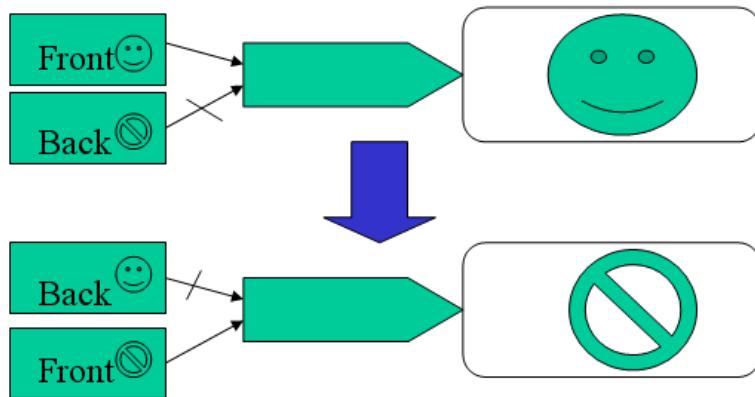


Рис. 2.5. Двойная буферизация

### 2.3. Что такое пиксель?

**Пиксель – это точка.**

Большая группа алгоритмов посвящена представлению геометрических фигур на растре: линий, многоугольников, эллипсов и т.д. Пиксель изображения рассматривается как нечто единое и элементарное. Геометрически в качестве пикселя рассматривается его центральная точка. Поскольку в этих задачах основным критерием является точность представления, то растр нередко изображают в виде квадратной (прямоугольной) сетки, узлы которой – это центры пикселей.

### 2.4. Осторожно – машинная точность

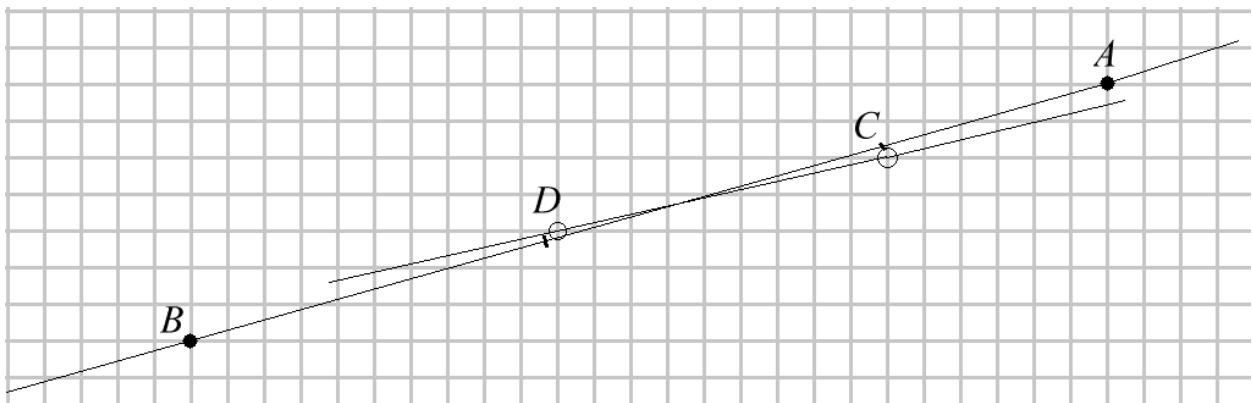


Рис. 2.6. Дискретная плоскость. Узлы сетки – представимые точки

При разработке алгоритмов вычислительной геометрии и компьютерной графики надо иметь в виду, что обычные факты из геометрии не всегда правильно работают. Для иллюстрации рассмотрим следующий пример. Обозначим через  $L(A, B)$  прямую на плоскости, проходящую через точки  $A$  и  $B$ . Классическая геометрия говорит, что если точки  $C$  и  $D$  таковы, что  $C \in L(A, B) \wedge D \in L(A, B)$ , то верно  $A \in L(C, D) \wedge B \in L(C, D)$ . Но в ЭВМ все вещественные числа представимы с некоторой точностью. Иначе говоря, все представимые в машине вещественные числа лежат в узлах некоторой сетки на числовой оси. Таким образом, все представимые в машине точки плоскости – это узлы некоторой прямоугольной сетки, возможно неравномерной, пусть так, как это показано на рис. 2.6. Используя уравнение прямой  $L(A, B)$ , мы будем считать, что нашли точки  $C$  и  $D$ , показанные рисками. Но в памяти машины мы запишем координаты точек, которые показаны кружками, т.е. ближайшие узлы машинной сетки. Теперь строим прямую  $L(C, D)$ . Оказывается, что точки  $A$  и  $B$  могут не лежать на этой прямой.

Очевидно, что еще большую опасность представляет использование экранных координат для промежуточных вычислений. Во многих задачах настоятельно рекомендуется переходить к экранной системе с целочисленными координатами только непосредственно перед выводом изображения.

## **2.5. Формат bmp-файла**

Для длительного хранения изображений используются файлы. Существует огромное количество форматов (или языков представления) для файлов изображений. За всеми ими закреплены уникальные расширения: "jpg", "bmp", "png", "gif", "tif", ... Как правило, появление нового многофункционального редактора изображений сопровождается появлением нового формата файла и уникального расширения, например, редактор PaintShopPro – "psp". Дело в том, что кроме самого изображения формат может хранить данные необходимые для самого редактора, такие как: уровневая структура изображения, история создания и т.д.

Наиболее простой формат – это формат "bmp" или битовая карта (bitmap). Изначально этот термин и формат применялись только для черно-белых изображений. На каждый пиксель отводился ровно один бит: 0 – черный пиксель, 1 – белый. В дальнейшем этот формат был расширен, в настоящее время он используется для хранения любых растрор: с палитрой и полноцветных. В рамках данного курса при выполнении упражнений нас будет интересовать только 24-хразрядный полноцветный формат "bmp". Он достаточно прост для программирования, все распространенные редакторы позволяют переводить изображения из любого формата в "bmp" и обратно.

Рассмотрим структуру и основные элементы bmp-файла (подробнее см. в MSDN) в объеме достаточном для выполнения задач в рамках данного курса.

Файл имеет двоичную структуру, все числа хранятся в формате big-endian (то есть младшие байты в начале, а старшие в конце). Так как процессоры на базе x86 используют такой же формат представления чисел, то программисту не потребуется их дополнительно преобразовывать при записи и чтении.

### **1. Заголовок файла.**

- Идентификация формата – 2 буквы – "BM". Несовпадение этой сигнатуры говорит о том, что файл испорчен или не содержит растра.
- Len (4 байта) – размер файла в байтах. Также позволяет распознать испорченные файлы.
- Резервные 4 байта.
- Start (4 байта) – номер байта, считая с нуля, с которого размещается информация о значениях пикселей раstra.

### **2. Информационный заголовок.**

- Длина информационного заголовка – число 40.
- H (4 байта) – высота раstra в пикселях.
- W (4 байта) – ширина раstra в пикселях.
- Число цветных плоскостей в изображении (2 байта). Полагаем всегда 1.
- Nbit (2 байта) – число битов значения для пикселя. Полагаем 24, поскольку в рамках курса мы будем работать только с такими изображениями.
- Применяемый метод сжатия информации (4 байта). Поскольку в рамках курса мы не будем использовать какой-либо компрессии данных, то полагаем 0.

- Nrastr (4 байта) – число байтов в информации о пикселях раstra. Очевидно, что существует четкая зависимость между значениями Len, Start, Hi, Wi, Nbit и Nrastr.

*Вопрос:* какая зависимость?

- Дальнейшая информация не имеет отношения к конкретному раstrу, а говорит, скорее, о том, в каких условиях изображение получалось: вертикальный и горизонтальный размеры экрана (по 4 байта).
- И не нужная нам информация о палитре, поскольку мы будем работать только с полноцветными изображениями. Эта информация включает: число используемых цветов (4 байта), число важных цветов (4 байта). Мы их полагаем равными 0.

**3. Палитра.** В нашем случае она отсутствует. В общем случае палитра содержит 256 входов по 4 байта: значение красного, значение зеленого, значение синего и неиспользуемый байт (полагается равным 0).

**4. Данные о пикселях раstra.** В файле они начинаются с байта с номером Start. Информация о пикселях записывается по строкам раstra, начиная с самой нижней. В строке – слева направо. Если значение пикселя равно (красный, зелёный, синий), то соответствующие байты записываются в обратном порядке: синий, зелёный, красный. Каждая строка раstra должна занимать в файле число байтов кратное 4-м. При необходимости для выравнивания добавляются нулевые байты.

Рассмотрим пример bmp-файла. Итак, пусть у нас в файле записано полноцветное (т.е. 24 бита на пиксель) изображение размером  $82 \times 66$ . Тогда начало файла в шестнадцатеричном представлении выглядит так:

00000000:	42 4D 46 40 00 00 00 00   00 00 36 00 00 00 28 00
00000010:	00 00 42 00 00 00 52 00   00 00 01 00 18 00 00 00
00000020:	00 00 10 40 00 00 12 0B   00 00 12 0B 00 00 00 00
00000030:	00 00 00 00 00 00 FF 80   00 FF 80 00 FF 80 00 FF
00000040:	80 00 FF 80 00 FF 80 00   FF 80 00 FF 80 00 FF 80
00000050:	00 FF 80 00 FF 80 00 FF   80 00 FF 80 00 FF 80 00
00000060:	FF 80 00 FF 80 00 FF 80   00 FF 80 00 FF 80 00 FF
00000070:	80 00 FF 80 00 FF 80 00   FF 80 00 FF 80 00 FF 80

42 4D – это буквы "BM". 46 40 00 00 надо читать как 00 00 40 46, что в десятичной системе 16454 – длина файла в байтах. 36 00 – это 00 36 => 54, адрес байта начала данных о пикселях. Сам этот пиксель (его цвет) помечен как FF 80 00. Далее, 28 00 = 28 => 40 – длина информационного заголовка. 42 00 00 00 = 42 => 66 – высота раstra в пикселях. 52 00 00 00 = 52 => 82 – ширина раstra. Nrastr = 10 40 00 00 (читать как 00 00 40 10) – это 16400. Проверяем  $54 + 16400 = 16454$ . Еще раз: в полноцветном изображении байты цвета записываются в обратном порядке, т.е. сначала синий, затем зелёный и красный: цвет FF 80 00 – это синий = 255, зелёный = 128, красный = 0.

### 3. Пиксельные области

*Пиксель – это область (квадрат).*

Растр или экранное изображение рассматриваются состоящими из одинаковых *прямоугольных* пикселей, например, первые дисплеи IBM/PC: CGA, MGA, EGA. Современные дисплеи имеют *квадратные* пиксели, и мы будем рассматривать только такие раstry. Понятие *области* подразумевает наличие связности между ее точками, имеется понятие границы. Поскольку растр – это дискретное множество пикселей, то эти понятия необходимо определить применительно к *пиксельным областям* (ПО).



Рис. 3.1. 4-связность и 8-связность

Связность точек области означает, что любые ее две точки можно соединить непрерывным путем, все точки которого также принадлежат этой же области. Для ПО введены следующие связности:

- **4-связность.** Путь строится из 4-х возможных элементарных движений (рис. 3.1,а);
- **8-связность.** Путь строится из 8-ми возможных элементарных движений (рис. 3.1,б).

Одна из задач состоит в том, чтобы определить все пиксели ПО, заданной определённым образом. Самое распространённое применение — это заливка (заполнение) данной ПО новым цветом.

Чтобы задать ПО, используются различные предикаты над значениями пикселей, например:

- Пиксель содержит ненулевую красную компоненту цвета.
- Рассматривая значение V для пикселя буфера кадра как целое число, можно говорить: " $V > 15 \ \&\& V < 55$ ".
- Расстояние в цветовом пространстве между цветом данного пикселя  $(r,g,b)$  и некоторым заданным цветом  $(r_0,g_0,b_0)$  с использованием нормы  $l_2$  не превышает C, то есть  $\sqrt{(r-r_0)^2 + (g-g_0)^2 + (b-b_0)^2} < C$ .

Наиболее часто используется простой предикат, истинный для пикселей какого-либо конкретного цвета, заданного вектором  $(r,g,b)$  или индексом палитры. В рамках данного раздела мы будем рассматривать только такие предикаты, что не ограничивает общности.

Пиксельные области бывают *внутренне определенные* и *гранично определенные*.

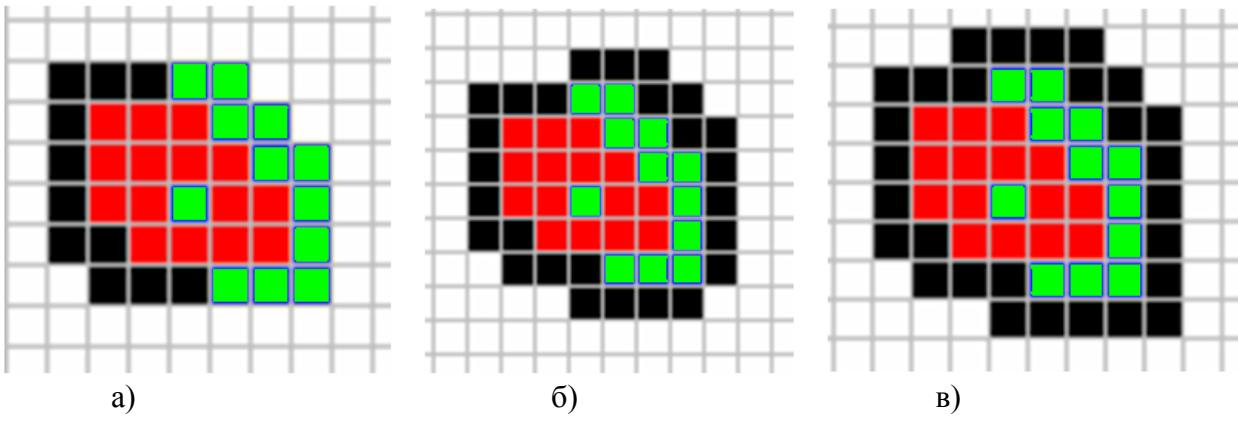


Рис. 3.2. а) внутренне определенная ПО; б и в) гранично определенные ПО

*Внутренне определенные ПО.* Связное (4 или 8) множество пикселей растра, которые удовлетворяют некоторому предикату, например, 4-х связная ПО, состоящая из пикселей красного цвета (255,0,0), как на рис. 3.2,а. Очевидно, что из себя представляет граница ПО. Особо отметим, что ПО дополнительно ограничены границами растра/изображения. Граница ПО может быть многосвязной в том случае, если область содержит дырки. На рис. 3.2.а видно, что граница ПО состоит из пикселей, имеющих различные значения, а сама ПО имеет дырку из одного пикселя.

На внутренне определенных ПО работают *внутренне заполняющие алгоритмы*. Например, перекраска пиксельной области цвета OldValue цветом NewValue выглядит следующим образом:

```
if (pixel == OldValue)
    pixel = NewValue
```

*Гранично определенные ПО.* Связное (4 или 8) множество пикселей растра, пиксели границы которого удовлетворяют некоторому предикату. Это определение говорит о том, что ни один пиксель ПО не должен удовлетворять этому предикату. Для простого предиката это определение можно записать следующим образом: "Гранично определенная ПО – это связное (4 или 8) множество пикселей растра, пиксели границы которого имеют одно и то же значение BoundaryValue". В этом случае граница также может рассматриваться как внутренне определенная пиксельная область. Отметим, что для 4-х связной ПО граница может быть 8-связной, а для 8-связной ПО граница должна быть 4-х связной. На рис. 3.2,б,в. показаны гранично определенные ПО.

Очевидно, что алгоритмы в зависимости от того, на какую связность они рассчитаны (4 или 8), могут давать различные результаты, поэтому при работе с редакторами изображений нужно внимательно заливать контурные рисунки – пиксельные области. Отметим, что если области задаются как 4-связные, то многие операции выполняются "как ожидаешь", да и с ними можно выполнять все операции как над областями.

Гранично заполняющие алгоритмы, как правило, каждому пикセルю области присваивают одно и то же значение NewValue. Многие реализации требуют, чтобы ни один пиксель границы области не имел значения NewValue.

*Вопрос:* как можно избежать этого ограничения?

### 3.1. Заполнение шаблоном

В рассмотренной выше задаче заполнения пиксельной области предполагалось, что после операции вся ПО будет закрашена единым новым цветом NewValue. Однако часто необходимо некоторую область заполнить повторяющимся рисунком – шаблоном. Например, на плане местности область лесопосадок заполнить обозначением хвойных или лиственных деревьев. Так сказать, простейшее текстурирование, когда считается, что

размеры пикселей раstra и шаблона подготовлены в едином масштабе, т.е. не надо пересчитывать изображение шаблона на другое разрешение.

Рассмотрим виртуальную раstralную плоскость  $P$ , точки которой  $(x, y)$  могут иметь целочисленные координаты от  $-\infty$  до  $+\infty$ . Ограниченнaя часть ее около начала координат – это наш растр (экран, рисунок):

$$R[N, M] = \{(x, y), x = 0..(N-1), y = 0..(M-1)\},$$

и в нем выделена некоторая пиксельная область  $O$ . Не вникая в детали ее задания, будем считать, что у нас есть характеристическая функция этой области –  $F_O(x, y)$  равная 1 для пикселей области и 0 для остальных. Шаблон – это тоже некоторый растр, как правило, другого (меньшего) размера –  $Pt[s, t] = \{(x, y), x = 0..(s-1), y = 0..(t-1)\}$ . Для выполнения данной операции необходимо определить точку привязки шаблона  $(x_r, y_r)$  на всей раstralной плоскости  $P$ . Заполнение шаблоном – это покрытие всей плоскости  $P$  шаблоном как кафельной плиткой. Как правило, рассматривается только пространство, ограниченное растром R.

Идея простейшего алгоритма может быть выражена следующим образом:

```

for (j = 0; j < M; j++) // = построчное сканирование растра
    for (i = 0; i < N; i++)
    {
        if (F_O(i, j) == 1) // принадлежит ли пиксель области?
        {
            // = определить координаты
            // "внутри своей копии кафельной плитки"
            is = (i - x0) % s;
            jt = (j - y0) % t;
            R[i][j] = Pt[is][jt]; // = переписать значение из
шаблона что-то я не понял тут, надо будет еще раз проверить
Деб
        }
    }
}

```

Во времена дефицита текстурной памяти учитывались такие характеристики шаблона, как симметрия его рисунка, что позволяло использовать в 2 или 4 раза меньшее описание.

### **3.2. Аффинные преобразования над пиксельными областями**

В качестве пиксельной области можно рассматривать раstralное представление любой фигуры: отрезок прямой, дуга окружности, буква и др. Довольно часто стоит задача применить к такой фигуре аффинное преобразование. Первая мысль, которая приходит в голову, дает следующий простой алгоритм.

Поскольку пиксельная область  $O$  является множеством пикселей  $\{(x_i, y_i)\}$ , то в качестве результата аффинного преобразования  $A$  считать множество пикселей раstra, в которые попадут образы центров пикселей области. Этот алгоритм действительно очень эффективен, если преобразование имеет простой вид типа поворота на 90 градусов или сдвига на вектор. Если преобразование включает поворот на произвольный угол (тем более – масштабирование с увеличением), то, применив его несколько раз даже к отрезку, мы можем потерять связность фигуры. В связи с этим необходимы более дорогие алгоритмы.

Пусть пиксельная область  $O$  имеет характеристическую функцию  $F_O(x, y)$ . Выше нам было достаточно определить ее только для целочисленных значений  $(x, y)$ . Доопределяем ее до

всего растра (или даже для всей вещественной плоскости) –  $F_{PO}(x, y)$ . Очевидно, что сам способ доопределения влияет на то, что мы будем понимать под областью на вещественной плоскости. Применяем следующее доопределение:

$$F_{PO}(x, y) = F_O([x + 0.5], [y + 0.5]), \quad (3.1)$$

где квадратные скобки означают взятие целой части. Такое определение говорит, что целочисленные координаты соответствуют центрам пикселей растра. Применение аффинного преобразования ко всем точкам области на вещественной плоскости является очень дорогой операцией. Вместо этого остановимся на следующем алгоритме.

Применяем к пиксельной области  $O$  аффинное преобразование  $A$ . Образом этого преобразования (результатом операции) будет пиксельная область  $AO$  с характеристической функцией  $F_{AO}(x, y) = F_{PO}(A^{-1}(x, y))$ . Другими словами каждый пиксель растра мы проверяем на попадание его в прообраз преобразования – в исходную пиксельную область  $O$ .

Так как аффинное преобразование легко представить в виде умножения на матрицу, найти обратное к нему также не представляет труда. Подробнее о преобразованиях в компьютерной графике см. главу 11.

### 3.3. Алгоритмы заливки пиксельной области

Простой пример растра на рис. 3.3. содержит три пиксельные области. Если в качестве границы возьмем наиболее черную из них, то непонятно границу какой области мы задали: белой или серой. Вспомним, что границы растра являются неявными границами ПО. Как правило, при работе с редактором пользователь указывает мышью на выбранную им область, т.е. задает один из принадлежащих ей пикселей (черные квадратики на рисунке). Такой пиксель называется затравкой (seed). Таким образом, разные пиксели-затравки могут определять различные ПО на одном и том же растровом изображении.

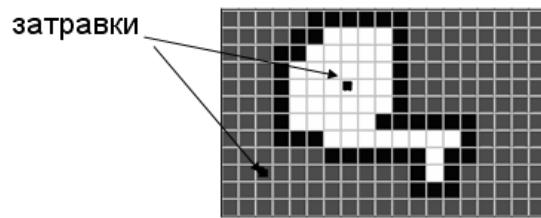


Рис. 3.3. Пример растра с пиксельными областями и затравок **ПЕРЕРИСОВАТЬ**

Рассмотрим очень простой рекурсивный алгоритм заливки с затравкой в пикселе  $(x, y)$ . Он должен перекрасить 4-связную ПО с пикселями цвета OldValue в цвет NewValue. ПО может иметь многосвязную границу.

```
FloodFill(x, y, OldValue, NewValue)
{
    if (Pixel(x, y) == OldValue)
    {
        Pixel(x, y) = NewValue;
        FloodFill(x + 1, y, OldValue, NewValue);
        FloodFill(x - 1, y, OldValue, NewValue);
        FloodFill(x, y + 1, OldValue, NewValue);
        FloodFill(x, y - 1, OldValue, NewValue);
    }
}
```

*Упражнение.* Преобразовать алгоритм для случая 8-связных ПО.

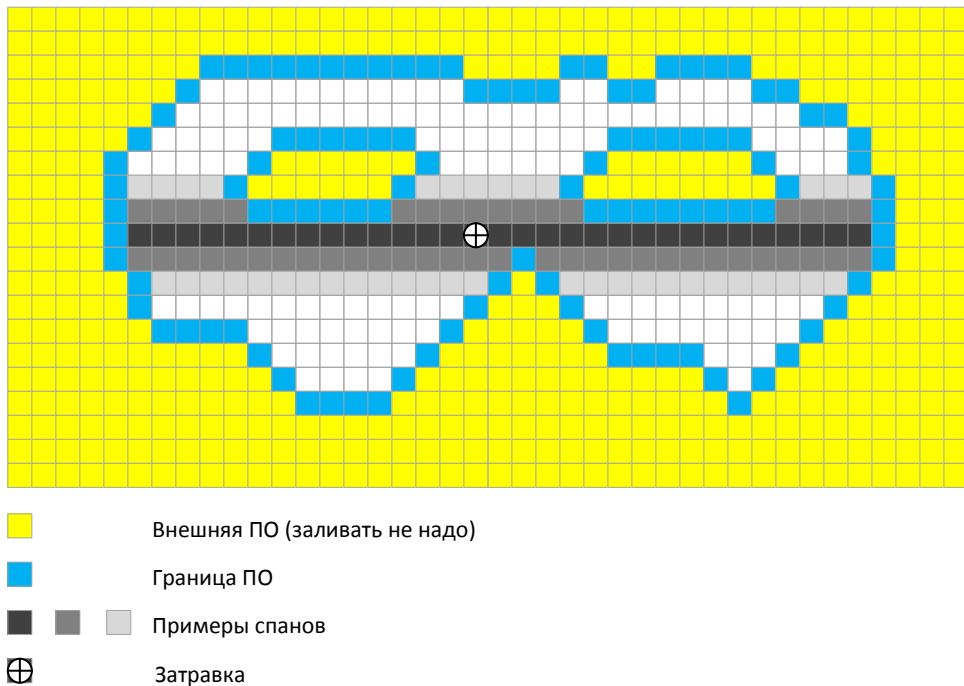


Рис. 3.4. Примеры спанов

Но простые алгоритмы работают, как правило, очень неэффективно. Попробуем представить большой растр (изображение для лазерного принтера) и сложную ПО. Сколько раз процедура будет вызвана рекурсивно, и какой объём стека ей потребуется?

Рассмотрим алгоритм заливки из семейства span filling algorithm. Он немного труднее в реализации, но зато работает более эффективно. Основное понятие – это спан (span) – непрерывный горизонтальный "отрезок" из пикселей, которые принадлежат области (имеют одно значение) и ограниченный с обоих концов пикселями, которые не принадлежат ей (имеют другие значения), см. рис. 3.4. На рисунке представлена пиксельная область с двумя дырками. Для примера приведены 11 спанов (один чёрный, пять тёмно-серых и пять светло-серых). Перейдем к формулировке алгоритма.

*Начало.* Заводим пустой стек. Определяем спан, содержащий затравку  $(x, y)$ , и помещаем его в стек. Очевидно, что  $\text{OldValue} = \text{Pixel}(x, y)$ .

*Основной цикл.*

**Пока** стек не пуст

- ```

{
    1. Берем из стека спан C.
    2. Заполняем его пиксели значением NewValue.
    3. Поднимаемся на 1 линию раstra вверх. Находим все спаны,
       связанные с C (согласно связности 4 или 8),
       и помещаем их в стек. Напомним: все пиксели спана
       содержат значение OldValue.
    4. Опускаемся на 1 линию раstra вниз. Находим все спаны,
       связанные с C (согласно связности 4 или 8),
       и помещаем их в стек.
}

```

## 4. Растеризация – приближение отрезков на растре

До 90-х годов прошлого столетия большую роль играли графопостроители, при помощи которых из ЭВМ выдавались графики на бумагу. Пищущий инструмент (перо, фреза) перемещался шаговыми двигателями и, таким образом, двигался по точкам некоторого квадратного растра. На первые графопостроители из ЭВМ подавались команды для выполнения каждого шага. При шаге 0.05 мм только рамка листа А4 210x290 мм требовала выдачи из ЭВМ 20 тыс. шагов. Это, конечно, было очень дорого, поскольку ЭВМ простила, ожидая пока команды будут переданы и отработаны. Тогда был введен режим off-line, при котором команды писались на магнитную ленту, а та в дальнейшем разрисовывалась при помощи автономного декодера, управляющего графическим устройством. Объем информации на магнитной ленте был огромным, а графический выход небольшой. Ситуация сильно изменилась, когда появились доступные 8-разрядные микропроцессоры, обладающие достаточно бедным набором команд, включающим операции целочисленного сложения и сдвига. Тогда обратили внимание на имеющую революционное значение статью Дж. Брезенхэма [2], вышедшую в 1965 году. В ней автор предложил алгоритм определения множества точек растра, которые составляют изображение отрезка. Этот алгоритм использовал только операции целочисленного сложения (вычитания) и умножения на 2, что эквивалентно операции сдвига. С этого времени графопостроители, усиленные микропроцессорами, получали из ЭВМ целиком отрезки, концы которых лежат на точках растра. Появилось много работ по данной тематике, а алгоритмы приближения на растре отрезков, окружностей, других алгебраических кривых получили название алгоритмов растеризации. Отметим, что мы до сих пор сталкиваемся с ними ежедневно, когда работаем за компьютером, поскольку каждый отрезок, который мы видим на экране, прошел через одну из модификаций алгоритма Брезенхэма.

*Растеризация* – это процесс преобразования вершинного (или алгебраического) описания фигуры в пиксельное представление. Растеризация часто называется *преобразованием сканирования*, поскольку многие алгоритмы растеризации применяют инкрементальные методы, сканируя фигуру по строкам (линиям) растра, эксплуатируя свойство *когерентности*. Инкрементальные методы вычисляют новые значения быстро на основе значений, полученных на предыдущем шаге, вместо того, чтобы вычислять новые значения напрямую, что часто оказывается довольно медленным процессом. Когерентность в пространстве или по времени – это термин, означающий, что близкие объекты (т.е. строки пикселей в нашем случае) имеют качественные характеристики подобные характеристикам текущего объекта.

Пусть у нас плоскость изображения  $P$  такова, что точки с целочисленными координатами являются центрами квадратных пикселей. Для пикселя  $p(i, j)$  множество всех его точек обозначим как  $D(p(i, j))$  или  $D(p)$ , а центральную точку –  $p^*(i, j)$  или  $p^*$ . Для определенности точки границы пикселя также отнесем к  $D(p)$ . Для пиксельной области  $O$  множество всех точек всех ее пикселей  $\bigcup_{\forall p \in O} D(p)$  обозначим через  $D(O)$ .

Рассмотрим одну из возможных постановок задачи растеризации алгебраических кривых. Под растеризацией (участка) непрерывной алгебраической кривой  $C$  будем понимать построение (или нахождение) такой пиксельной области  $PO(C)$ , которая удовлетворяет ряду условий, например:

- $C$  целиком лежит в  $D(PO(C))$ ;
- $PO(C)$  имеет связность 4.

Совершенно аналогично можно определить растеризацию любой плоской геометрической фигуры. На рис. 4.1 слева приводится пример,  $PO(C)$  – это закрашенные пиксели – для кривой  $C$ . Обратите внимание на правую часть рис. 4.1. Здесь мы тоже видим растровое

представление кривой  $C$  в виде пиксельной области, но эта ПО 8-связная. И есть точки кривой, которые не принадлежат  $PO(C)$ .

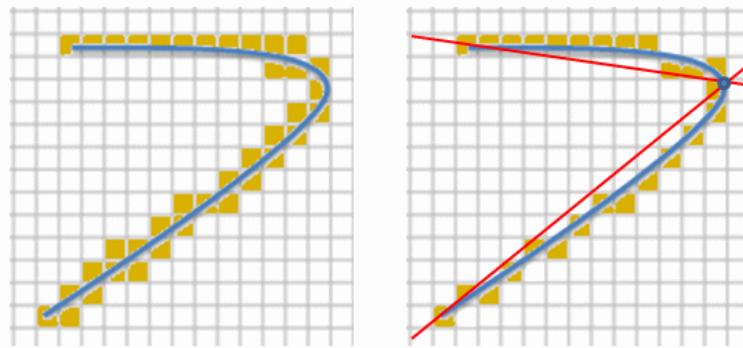


Рис. 4.1. Растеризация кривой 4-связной ПО и аппроксимация 8-связной ПО

В ряде случаев более экономное представление оказывается вполне достаточным (см. следующий пункт). Это аппроксимация кривой при помощи 8-связной ПО –  $PO_8(C)$ . Каким условиям должна удовлетворять эта область кроме 8-связности? Можно предложить эмпирическое правило: добавляемый пиксель (его центр) должен быть ближе к кривой, чем все уже построенные пиксели (их центры). Если пытаться навести полный формализм, то можно уйти в дебри, связанные с кривизной кривой и т.п. Как правило, растровые аппроксимации строятся для участка кривой, который можно представить как график однозначной монотонной кривой относительно некоторой прямой. На рис. 4.1 жирная точка показывает разбиение кривой на 2 участка, а тонкие линии – те самые "некоторые" прямые. В окрестности стыковок таких участков надо делать дополнительные решения.

#### 4.1. Симметричность

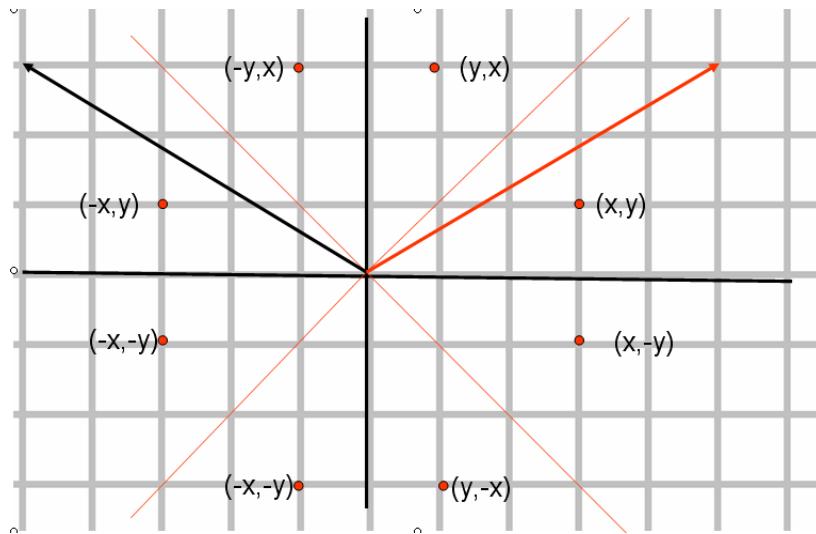


Рис. 4.2. 8-симметрия. Узлы сетки – это центры пикселей

Для того чтобы растеризовать произвольный отрезок с целочисленными координатами  $[(x_1, y_1), (x_2, y_2)]$  достаточно уметь растеризовывать отрезки  $[(0,0), (dx, dy)]$  ( $dx \geq dy \geq 0$ ), начинающиеся в начале координат и лежащие в секторе координатной плоскости с углом в 45 градусов, см. рис. 4.2. Различных секторов плоскости 8 штук. Из рисунка видно, какое преобразование координат надо делать, чтобы привести исходный отрезок, начинающийся в начале координат, в отрезок, лежащий в первом секторе, а затем как перевести полученные точки растра обратно в исходный сектор. Сдвигаем отрезок  $[(x_1, y_1), (x_2, y_2)]$  в начало

координат, а затем, пользуясь 8-симметрией, приводим его к виду  $[(0,0), (dx, dy)]$ ,  $dx \geq dy \geq 0$ .

Задача растеризации значительно упростилась.

## 4.2. Наивный алгоритм

Исходные данные: отрезок  $S = [(0,0), (dx, dy)]$ , целые координаты  $dx \geq dy \geq 0$ .

Построить 8-связную пиксельную область  $PO(S)$ , аппроксимирующую  $S$ .

В первом секторе отрезки описываются уравнением

$$y = mx, 0 \leq m \leq 1, m = \frac{dy}{dx}.$$

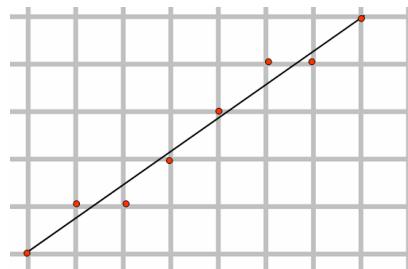


Рис. 4.3. Аппроксимация отрезка точками раstra

Очевидно, что раcтровая аппроксимация отрезка имеет ровно одну точку на каждой вертикали. С учетом этого мы можем записать следующий алгоритм:

$$x_0 = 0, y_0 = 0; \begin{cases} x_{i+1} = x_i + 1, \\ y_{i+1} = y_i + m. \end{cases}$$

Получив последовательность точек плоскости  $\{(x_i, y_i)\}_{i=0}^{dx}$ , округляем вторые координаты для получения точек раstra и получаем искомое представление отрезка. Что мы затрачиваем на каждую точку? 2 вещественных сложения, 1 вещественное сложение для округления и операцию перевода из вещественного в целое.

Обычно подобные алгоритмы называются *цифровыми дифференциальными анализаторами* (digital differential analyzer – DDA). А теперь посмотрим, как предложил решать такие задачи Брезенхэм.

## 4.3. Алгоритм Брезенхэма для отрезков

В литературе имеется несколько выводов этого алгоритма. Часто используется алгебраический подход, но мы используем геометрическую интерпретацию. Уравнение прямой отрезка имеет вид:  $y = \frac{dy}{dx} x$ .

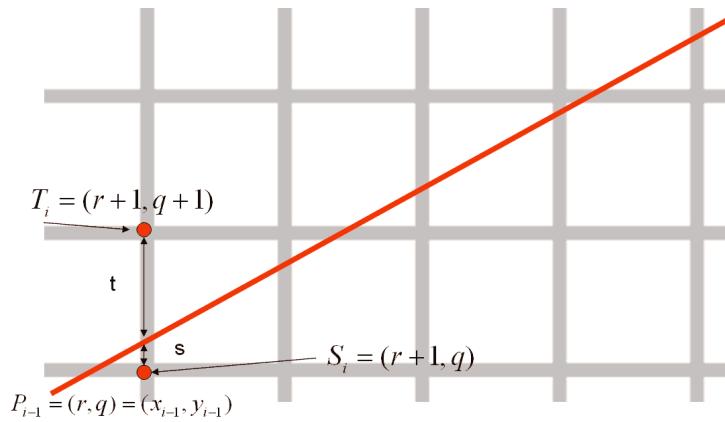


Рис. 4.4. Выбор очередной точки раstra

Начальная 0-я точка растрового отрезка известна – это  $(0,0)$ . Пусть мы уже построили  $i$  точек, последняя  $P_{i-1} = (r, q) = (x_{i-1}, y_{i-1})$ . Стоит задача выбора следующей точки, см. рис. 4.4. Горизонтальная координата очевидна  $x_i = x_{i-1} + 1$ . В качестве очередной точки раstra мы можем взять либо точку  $T_i = (r+1, q+1)$ , либо точку  $S_i = (r+1, q)$ . Основная идея Брезенхэма заключается не в том, чтобы делать вычисления напрямую, а в том, чтобы найти дешевый с вычислительной точки зрения *критерий выбора* между  $T_i$  и  $S_i$ . Введем две величины:

$$s = \frac{dy}{dx}(r+1) - q, \quad t = (q+1) - \frac{dy}{dx}(r+1).$$

Вычислим разность  $s - t = 2\frac{dy}{dx}(r+1) - 2q - 1$ . Если это выражение неотрицательное, то мы выбираем точку  $T_i$ , иначе – точку  $S_i$ . Значит, знак этой разности уже можно использовать как критерий. Но операций не стало меньше, а вычисления даже усложнились. Избавимся от знаменателя (он положительный) и введем величину *критерия*:

$$d_i = dx(s - t) = 2(r \cdot dy - q \cdot dx) + 2dx - dy.$$

Подставим  $r$  и  $q$  и запишем критерий для двух последовательных шагов:

$$\begin{aligned} d_i &= 2x_{i-1}dy - 2y_{i-1} + 2dx - dy, \\ d_{i+1} &= 2x_i dy - 2y_i + 2dx - dy. \end{aligned}$$

Вычислим разность

$$d_{i+1} - d_i = 2dy(x_i - x_{i-1}) - 2dx(y_i - y_{i-1}).$$

Теперь осталось записать весь алгоритм:

*Начало.* Точка  $(0,0)$ ,  $i = 0$ ,  $d_1 = 2dy - dx$ .

*Шаг*  $i = 1, \dots, dx$ .

$$x_i = x_{i-1} + 1;$$

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

Если  $d_i < 0$ , то {выбираем  $P_i = S_i$ ,  $y_i = y_{i-1}$ ,  $d_{i+1} = d_i + 2dy$ }

Таким образом, для вычисления  $d_{i+1}$  достаточно вычислительного устройства, которое позволяет выполнять целочисленное сложение и сдвиг, при помощи которого реализуется умножение на 2.

*Упражнение.* Обосновать значение  $d_1$ .

#### 4.4. Алгоритм Брезенхэма для окружности

Любую окружность мы также сдвигаем в начало координат, там получаем растровый образ, а в конце скорректируем полученные координаты на первоначальный сдвиг.

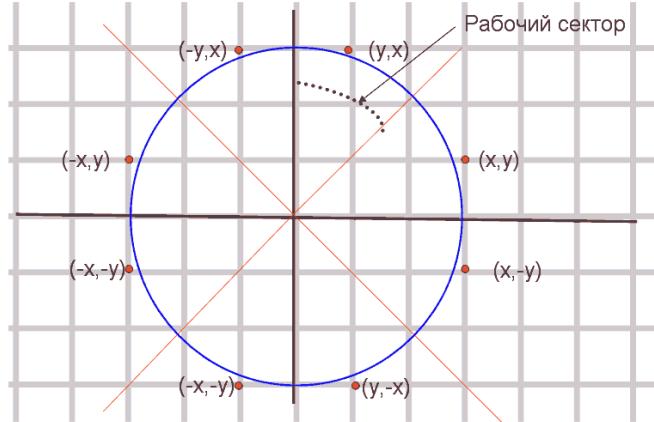


Рис. 4.5. 8-симметрия для окружности

Используем 8-симметрию, как показано на рис. 4.5. Получая одну точку растрового приближения в рабочем секторе, мы размножаем ее семикратно. Особые случаи:  $x = 0, y = r$  – эта точка дублируется только один раз; если получается точка  $x = y$ , то она дублируется трижды. Рабочий сектор выбран так, как показано на рисунке, поскольку в данном случае все точки растрового представления дуги будут иметь различные горизонтальные координаты.

*Упражнение.* Обоснуйте последнее утверждение.

Окружность имеет уравнение  $x^2 + y^2 = r^2$ , очевидно, что  $x, y, r$  не могут быть все целые, т.е. сразу указывать на центры пикселей. На основе прямых вычислений наивный алгоритм построения 1/8 окружности имеет следующий вид:

*Начало.*  $x_0 = 0, y_0 = r$ .

*Шаг.* Цикл  $\{x_i = x_{i-1} + 1; y_i = \sqrt{r^2 - x_i^2};$  если  $(x_i < y_i)$  ВЫХОД;  $y_i = [y + 0.5]\}$

Очевидно, что он содержит серьезные арифметические операции.

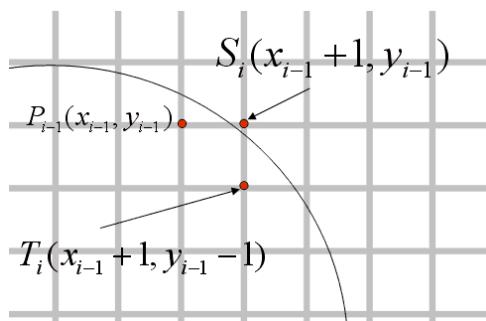


Рис. 4.6. Схема выбора очередной точки растра

Брезенхэм предложил алгоритм и для построения растрового образа окружности. Вспомогательное геометрическое построение на рис. 4.6. выглядит подобно предыдущему, см. рис. 4.4. Мы не будем здесь повторять выкладки, идея которых напоминает идею, примененную в п. 4.2, также см. [8]. Сразу запишем алгоритм.

*Исходные данные.* Центр окружности – точка  $(0,0)$ , радиус окружности – целое  $r$ .

*Начало.*  $x_0 = 0, y_0 = r, d_1 = 3 - 2r$ .

*Шаг.* Если  $d_i < 0$  {выбираем  $P_i = S_i; x_i = x_{i-1} + 1; d_{i+1} = d_i + 4x_{i-1} + 6$ }

Если  $d_i \geq 0$  {выбираем  $P_i = T_i; x_i = x_{i-1} + 1; d_{i+1} = d_i + 4(x_{i-1} - y_{i-1}) + 10$ }

После работ Брезенхэма было очень много других работ, посвященных как растеризации отрезков, так и более сложных алгебраических кривых.

#### 4.5. Необходимость математической модели

Пусть кривая задана некоторым уравнением  $f(x, y) = 0$ . Это ее математическая модель. Для того чтобы получить изображение кривой необходимо рассчитать ее растровое представление. Выполняя различные аффинные преобразования над конструкцией, в которую входит данная кривая, необходимо различать: выполняется преобразование растрового представления конструкции или преобразование самой конструкции. В последнем случае после каждого преобразования необходимо заново выполнять растеризацию кривой.

Аналогичное замечание относится и к областям. Не вникая в детали задания области  $O$  будем считать, что у нас есть характеристическая функция этой области —  $F_O(x, y)$ . Таким образом, мы можем задавать "независимые от раstra" фигуры, которые будут иметь всегда гладкие границы даже при многократном увеличении. Если же характеристическая функция области строится на основе пиксельной связности, то большей информации из этого (в общем случае) не получишь.

### 5. Многоугольники

Многоугольники (или полигоны) играют очень важную роль в компьютерной графике, поэтому мы должны внимательно с ними разобраться. Для начала рассмотрим два определения многоугольников из математического энциклопедического словаря [9].

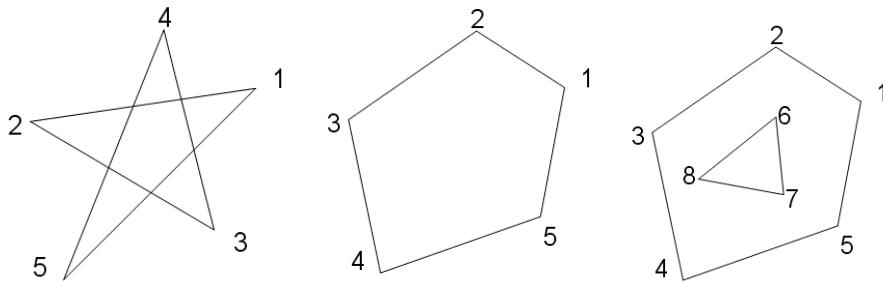


Рис. 5.1. Примеры фигур (многоугольники или нет?)

*Определение О1.* Многоугольник — замкнутая ломаная линия на плоскости, которая получается, если взять  $n$  любых точек  $A_1, \dots, A_n$  и соединить прямолинейным отрезком каждую из них с предыдущей, а последнюю — с первой:  $[A_1, A_2], \dots, [A_n, A_1]$ . Определенный таким образом многоугольник допускает самопересечение границы. Другими словами, многоугольник — это некоторый замкнутый путь, а отрезки называются сторонами многоугольника. Под это определение подходят две (в центре и слева) фигуры на рис. 5.1.

*Определение О2.* Многоугольник — это связная часть плоскости, вся граница которой состоит из конечного числа отрезков, называемых сторонами многоугольника. Могут быть и неограниченные многоугольники, т.е. часть плоскости, ограниченная конечным числом отрезков и полупрямых. Под это определение подходят также две (в центре и справа) фигуры на рис. 5.1. Договоримся, что мы будем опираться на определение О2. Чтобы многоугольник удовлетворял определению области, потребуем, чтобы он был открытым множеством.

Примером неограниченного многоугольника является плоскость с многоугольной "дыркой". Для того чтобы сказать, что имеется в виду под многоугольником (случай рис.5.1, в центре):

неограниченная плоскость с дыркой или ограниченная область; необходима дополнительная информация. Часто для этого достаточно знать: принадлежит области или не принадлежит бесконечно удаленная точка. В том же словаре [9] мы находим определение, которое однозначно решает данную проблему выбора.

*Определение О3.* Ориентированный многоугольник. Если каждой стороне многоугольника приписать направление и все стороны упорядочить, т.е. создать из них ориентированный путь, то положительной считается ориентация, когда обход границы многоугольника (контура) делается против часовой стрелки – область остается слева. Такое определение ориентации верно и для области, не только для многоугольника. Возможны многосвязные границы – см. рис. 5.1, справа – обход границы осуществляется согласно нумерации вершин.

В литературе часто встречаются публикации, когда положительным считается обход, при котором область остается справа. Поэтому при разборе алгоритмов надо сначала выяснить используемые определения многоугольника и положительного обхода его границы.

## 5.1. Характеристическая функция

Определение принадлежности точки плоскости многоугольнику выполняется согласно правилу чет-нечет (even-odd). Из тестируемой точки плоскости выпускается луч в произвольном направлении и подсчитывается число его пересечений со сторонами (с границей) многоугольника. Если полученное число пересечений четное (0, 2, 4, ...), точка лежит вне многоугольника, а если нечетное, точка принадлежит многоугольнику. На рис. 5.2. точки 1 и 2 не принадлежат многоугольнику, а точка 3 принадлежит.

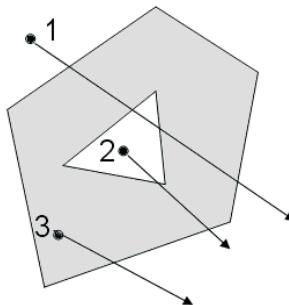


Рис. 5.2. Правило чет-нечет

В связи с замечанием о машинной точности (см. п. 2.3) могут возникать различные вычислительные проблемы:

1. Тестируемая точка лежит на границе. В зависимости от решаемой задачи ее можно считать пересечением. Это требуется специально оговаривать.
2. Луч проходит через вершину.

Данные случаи лучше исключать, для чего в предыдущей точке пересечения луча с границей (или в самой тестируемой точке) нужно повернуть луч на некоторый угол, т.к. вместо луча можно использовать любую кривую или ломаную, заканчивающуюся в бесконечности.

Итак, характеристическая функция многоугольника  $M$ :

$$\chi_M(x, y) = \begin{cases} 0, & \text{если } (x, y) \notin M \\ 1, & \text{если } (x, y) \in M \end{cases}$$

Погрешность при вычислениях может приводить к ошибкам для приграничных точек. В ряде приложений используется следующая функция:

$$\tilde{\chi}_M(x, y) = \begin{cases} \delta, & \text{если } (x, y) \in \overset{\circ}{M} \\ 0, & \text{если } (x, y) \in \partial M, \\ -\delta, & \text{если } (x, y) \notin M \end{cases}$$

где  $\delta$  – это расстояние от точки до границы, а  $\partial M$  — граница. Такая функция может оказаться значительно удобнее: к примеру, пользуясь только ей, можно от любой точки плоскости легко прийти к границе многоугольника.

Для того чтобы завершить разбор понятия многоугольника, рассмотрим еще одно определение.

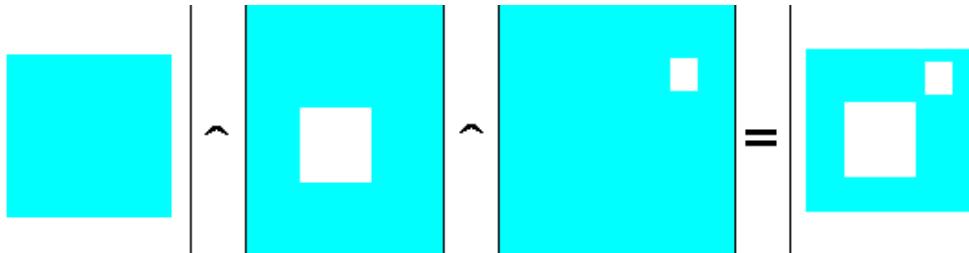


Рис. 5.3. Пример конструктивного определения многоугольника

*Определение О4.* Многоугольник  $M$  по определению О2 может быть представлен в виде пересечения  $M = \bigcap_{i=1}^n M_i$  конечного числа связных (ограниченных или неограниченных) многоугольников  $M_i, i = 1..n$ , имеющих односвязную границу. Пример дан на рис. 5.3, где показано пересечение ограниченного и двух неограниченных многоугольников.

Для того чтобы результаты ряда операций не выводили из класса объектов, отнесем к многоугольникам две области специального вида:  $\emptyset$  (пустой) и  $R^2$  (вся плоскость).

В компьютерной графике рассматриваются как выпуклые многоугольники, так и невыпуклые. Многоугольники с самопересечением границ используются редко по очевидной причине, т.к. они являются объединением более простых.

Наиболее популярные задачи с многоугольниками на плоскости:

- Отсечение или клиппирование (clipping) многоугольников.
- Растровая развертка или в более общей постановке – закраска многоугольников (shading).

## 5.2. Отсечение многоугольников

Отсечение – это необходимая операция, поскольку самое обычное ее применение – отображение на прямоугольном экране части многоугольника, попадающей в него. На самом деле отсечение рассматривается более широко – отсечение по многоугольнику (по любой фигуре). Случай, когда многоугольник является прямоугольным окном, наиболее важен.

Алгоритмы отсечения можно разделить:

- а) отсечение отрезков и линий более высокого порядка;
- б) отсечение пути "пера";
- в) отсечение многоугольников и других фигур.

Предположим, нам необходимо растеризовать отрезок, заданный координатами концов, или многоугольник, заданный множеством вершин. Наивный подход заключается в применении алгоритма растеризации (к примеру, алгоритма Брезенхема для отрезка), а затем определении, попал ли определенный пиксель на экран. Однако значительная часть фигуры

или вся она может оказаться вне экрана, при этом растровое представление может содержать тысячи, а то и миллионы пикселей. Тогда мы потратим массу машинного времени с нулевым видимым результатом. Экономия времени вычислений требует отбрасывать незначащие части как можно раньше. В связи с этим алгоритмы клиппирования разрабатывались, начиная с самых первых рисующих программ.

Если решаемая задача предполагает, что многие фигуры могут не попасть на экран вообще, можно использовать проверку габаритных боксов. Габаритный бокс — это минимальный прямоугольник со сторонами, параллельными осям координат, полностью содержащий искомую фигуру. Как нетрудно убедиться, для отрезка его концы будут совпадать с противоположными углами габаритного бокса, а для конечного многоугольника углы бокса определяются через минимумы и максимумы координат всех вершин. То есть, если углы бокса —  $(u_1, v_1)$  и  $(u_2, v_2)$ , а вершины многоугольника заданы как  $(x_i, y_i), i=1..n$ , то

$$u_1 = \min x_i; v_1 = \min y_i$$

$$u_2 = \max x_i; v_2 = \max y_i$$

Иногда может потребоваться расширить габаритный бокс, чтобы учесть толщину границ. Зная габаритный бокс и размеры экрана  $(a, b) - (c, d)$ , мы можем проверить, пересекается ли бокс с экраном. В этом случае истинен следующий предикат (частный случай теоремы о разделяющей оси [Здесь бы ссылку]):

$$(u_2 > a) \& (u_1 < c) \& (v_2 > b) \& (v_1 < d).$$

Если габаритный бокс не пересекается с экраном, то и фигура тоже не пересекается, а значит её можно не растеризовывать. Обратное же не всегда верно: фигура может лежать целиком вне экрана, но её габаритный бокс пересечётся с экраном. Таким образом ясно, что хотя проверка габаритного бокса даст заметный прирост производительности, она не решит все проблемы.

Далее мы рассмотрим методы, позволяющие не только точно определить, пересекается ли многоугольник с экраном, но и получить усечённый многоугольник. С историей разработки этих алгоритмов можно ознакомиться в литературе, а здесь мы остановимся на двух наиболее значимых алгоритмах, которые почти повсеместно используются и в наше время.

### **5.3. Алгоритм Сазерленда-Ходжмана**

Данный алгоритм предназначен для отсечения пути пера (ломаной) или не обязательно выпуклого многоугольника  $M$  по выпуклому многоугольнику-окну  $W$ . В результате отсечения может получиться: пустой многоугольник —  $\emptyset$ , один многоугольник, более одного многоугольника. Отметим, что данный алгоритм легко распространяется на отсечение многоугольников в пространстве выпуклым многогранником, в том числе неограниченным (скажем, пирамидой видимости).

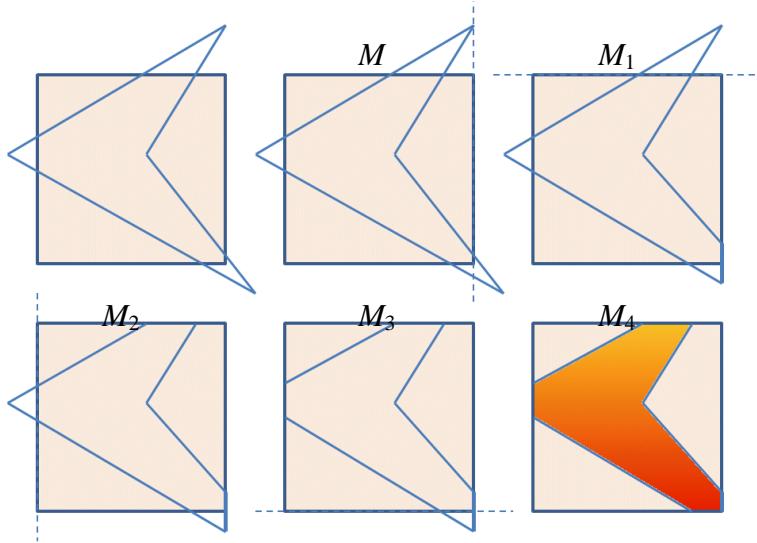


Рис. 5.4. Идея алгоритма

Рассмотрим случай, когда  $M$  имеет односвязную границу. Пусть он задан последовательностью своих вершин  $V_0, V_1, \dots, V_{n-1}$ . Многоугольник  $W$  выпуклый, и мы используем это его свойство, а именно: весь многоугольник  $W$  лежит по одну сторону от прямой, которая является продолжением любой из сторон. Эти прямые мы будем называть ограничениями. В примере взято прямоугольное окно  $W = [a, b] \times [c, d]$ . Проиллюстрируем работу алгоритма серией рисунков, показанных на рис. 5.4. Пунктиром показаны ограничения – прямые, являющиеся продолжениями сторон окна.

1. Исходная позиция: прямоугольное окно и отсекаемый многоугольник  $M$ .
2. Взяли ограничение  $x = b$  и прогнали все вершины  $M$ . В результате часть  $M$ , выходящая за пределы окна, оказалась отсеченной, вершины  $M$  скорректированы. Получили многоугольник  $M_1$ .
3. Взяли ограничение  $y = d$  и прогнали все вершины  $M_1$ . В результате часть  $M_1$ , выходящая за пределы окна, оказалась отсеченной, вершины  $M_1$  скорректированы. Получили многоугольник  $M_2$ .
4. Взяли ограничение  $x = a$  и прогнали все вершины  $M_2$ . В результате часть  $M_2$ , выходящая за пределы окна, оказалась отсеченной, вершины  $M_2$  скорректированы. Получили многоугольник  $M_3$ .
5. Взяли ограничение  $y = c$  и прогнали все вершины  $M_3$ . В результате часть  $M_3$ , выходящая за пределы окна, оказалась отсеченной, вершины  $M_3$  скорректированы. Получили многоугольник  $M_4$ . Это и есть результат, показанный на последнем рисунке.

Прежде, чем перейти к алгоритму рассмотрим вспомогательную операцию. Напомним, что границы многоугольников заданы с положительным обходом, когда его внутренность остается слева.

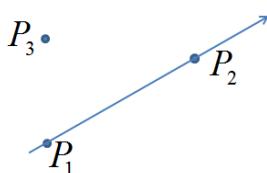


Рис. 5.5. Положение точки относительно ориентированной прямой

Рассмотрим рис. 5.5. С какой стороны по отношению к ограничению  $[P_1, P_2]$  лежит точка  $P_3$ ?

Вычислим координату  $z$  векторного произведения  $[P_1, P_2] \times [P_1, P_3]$ :

$$z = (x_3 - x_1)(y_2 - y_1) - (x_2 - x_1)(y_3 - y_1).$$

Имеем три случая:  $z > 0$  – точка  $P_3$  лежит справа от ограничения;  $z = 0$  – точка  $P_3$  лежит на ограничении;  $z < 0$  – точка  $P_3$  лежит слева от ограничения.

Теперь рассмотрим **все** ситуации, которые могут при обработке очередной вершины отсекаемого многоугольника, опираясь на рис. 5.6. В алгоритме последовательно рассматриваются отрезки – стороны отсекаемого  $M$ . Пусть  $[s, p]$  очередной отрезок. По отношению к очередному ограничению окна (полупространство слева от ограничения закрашено, а значит внутри окна) возможны следующие четыре случая:

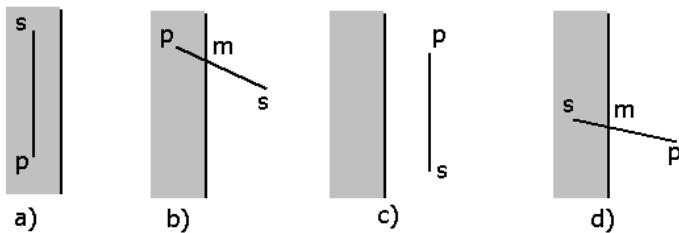


Рис. 5.6. Возможные ситуации

- a) отрезок полностью лежит слева от ограничения, т.е. не отсекается этим ограничением;
- b) отрезок входит в окно,  $m$  – точка входа;
- c) отрезок полностью лежит в правом полупространстве – вне окна;
- d) отрезок выходит из окна,  $m$  – точка выхода.

*Начало алгоритма.* Отсекаемый многоугольник  $M$ , вершины  $V_0, V_1, \dots, V_{n-1}$  заданы с обходом против часовой стрелки, отрезки  $[V_0, V_1], \dots, [V_{n-2}, V_{n-1}], [V_{n-1}, V_0]$ .

Окно  $W$  – выпуклый многоугольник, вершины  $W_0, W_1, \dots, W_{q-1}$  заданы с обходом против часовой стрелки, отрезки  $[W_0, W_1], \dots, [W_{q-2}, W_{q-1}], [W_{q-1}, W_0]$ .

*Цикл по ограничениям  $k = 0..q-1$*

{

*Цикл по сторонам  $M$   $j = 0..n-1$*

{

*Шаг 1.* Если  $j = n-1$  то  $[s, p] = [V_j, V_0]$ , иначе  $[s, p] = [V_j, V_{j+1}]$ .

*Шаг 2.* В зависимости от ситуации взаиморасположения отрезка и ограничения

выполняем согласно рис. 5.5 следующие действия:

- a) в выходной список вершин помещаем  $p$ , т.к.  $s$  была рассмотрена на предыдущем шаге по  $j$ .
- b) определяем  $m$ , выдаем на выход  $m$ .
- c) никаких действий, здесь мы уменьшили число вершин;
- d) определяем  $m$ , выдаем на выход  $m$  и  $p$  – две вершины. Здесь мы увеличили число вершин:

} // конец цикла по сторонам  $M$

} // конец цикла по ограничениям  $W$

- Вопросы и упражнения.** 1. Когда происходит обработка вершины  $V_0$ ? Не стоит забывать, что у многоугольника все вершины равноправны.  
 2. Как модифицировать алгоритм, чтобы отсекать путь пера (ломаную), а не многоугольник?  
 3. Построить и объяснить примеры, когда на выходе получается более одного многоугольника. Ни одного многоугольника.

#### 5.4. Алгоритм Вейлера-Азерттона

Предыдущий алгоритм используется достаточно широко в конвейере визуализации, т.к. экранные окна, как правило, выпуклые, а для рендеринга трехмерных сцен также применяются выпуклые пирамиды видимости. Но нередки случаи, когда окно является невыпуклым многоугольником, например, для показа различных детализаций на картах и чертежах. Для решения задачи отсечения по невыпуклому окну-многоугольнику рассмотрим простой, но мощный алгоритм Вейлера-Азерттона для случая, когда отсекаемый многоугольник  $M$  и отсекающий  $W$  имеют односвязную границу. Они могут быть ограниченными и неограниченными. В результате отсечения могут получиться: пустой многоугольник  $\emptyset$ , один или более одного многоугольника.

Идея самого алгоритма и применяемые правила очень простые.  $M$  отсекается по границам  $W$ , путем отслеживания в направлении против часовой стрелки границы  $M$ , вплоть до ее пересечения с границей  $W$ .

- Если ребро  $M$  входит в  $W$ , то обход идет вдоль ребра  $M$ .
- Если ребро  $M$  выходит из  $W$ , то производим поворот налево и идем вдоль ребер  $W$ .

*Замечание.* В случае другого обхода границы все правила записываются наоборот: "налево" заменяется на "направо", а "входит" и "выходит" поменять местами.

Пример на рис. 5.7 демонстрирует работу алгоритма. Хотя на приведенном примере  $W$  выпуклый (прямоугольник), мы этим его свойством не пользуемся.

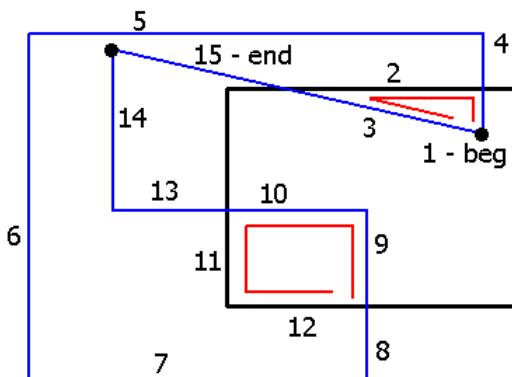


Рис. 5.7. Пример работы алгоритма

Теперь рассмотрим его работу более детально. Предполагается, что каждый из многоугольников задан списком вершин, причем таким образом, что при движении по списку вершин в порядке их задания внутренность многоугольника находится слева от границы. В случае пересечения границ отсекаемого многоугольника и окна возникают точки двух типов:

- Входные точки, когда ориентированное ребро отсекаемого многоугольника входит в окно,
- Выходные точки, когда ребро отсекаемого многоугольника идет с внутренней на внешнюю сторону окна.

Распишем последовательность действий.

- Строятся списки вершин отсекаемого многоугольника  $LM = \{V_0, V_1, \dots, V_{n-1}\}$  и окна  $LW = \{W_0, W_1, \dots, W_{q-1}\}$ .

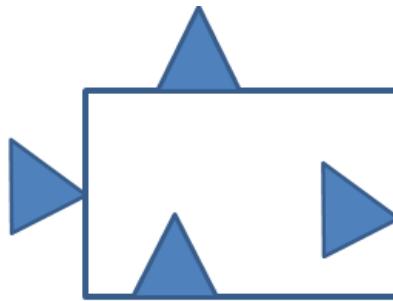


Рис. 5.8. Случаи, не считающиеся пересечениями

- Отыскиваются все точки пересечения. При этом расчете касание не считается пересечением, т.е. когда вершина или ребро отсекаемого многоугольника инцидентны или сторона М частично совпадает со стороной окна, см. рис. 5.8. Действительно, отсекать нечего. На рис. 5.9, слева случаи 1–4 учитываются как пересечения, а случаи 5 и 6 – нет. Справа показаны точки 5 и 6 немного оттянутые влево на некоторое расстояние  $d > 0$ . Тогда левый рисунок можно рассматривать как предельный случай при  $d \rightarrow 0$ . Конечно, в качестве пересечений можно учитывать случаи 5 и 6 вместо случаев 3 и 4.

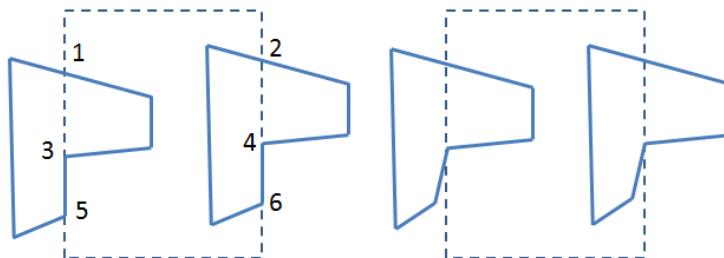


Рис. 5.9. Частные случаи пересечений

- Списки координат вершин отсекаемого многоугольника и окна дополняются новыми вершинами с координатами точек пересечения. Причем, если точка пересечения  $P_k$  находится на ребре  $[V_i, V_j]$ , то последовательность точек  $V_i, V_j$  превращается в последовательность  $V_i, P_k, V_j$ . Возможны даже более сложные модификации списков  $LM$  и  $LW$ , поскольку новое пересечение  $P_l$  может быть обнаружено на ребре  $[V_i, P_k]$ . В этом случае последовательность вершин превратится в  $V_i, P_l, P_k, V_j$ . Если на одном ребре несколько пересечений, они должны быть в правильном порядке:  $P_l$  ближе к  $V_i$ , а  $P_k$  — к  $V_j$ . При этом устанавливаются двусторонние связи между одноименными точками пересечения в списках вершин отсекаемого многоугольника и окна. Из входных и выходных точек пересечения образуются отдельные подсписки входных и выходных точек в списках вершин.
- Далее выполняется построение результата, который может состоять из нескольких многоугольников. Или ни одного, когда список входных точек пуст.
  - Начинаем строить первый многоугольник результата, для чего берем первую точку в списке входных точек.

- Двигаемся по вершинам отсекаемого многоугольника пока не обнаружится следующая точка пересечения; все пройденные точки, не включая прервавшую просмотр, заносим в результат.
- Используя двухстороннюю связь точек пересечения, переключаемся на просмотр списка вершин окна. Двигаемся по вершинам окна до обнаружения следующей точки пересечения; все пройденные точки, не включая последнюю, прервавшую просмотр, заносим в результат.
- Используя двухстороннюю связь точек пересечения, переключаемся на список вершин обрабатываемого многоугольника.
- И т.д. Эти действия повторяем, пока не будет достигнута исходная вершина. Это означает, что очередная часть отсекаемого многоугольника, попавшая в окно, замкнулась.

5. Если не исчерпан список входных точек пересечения, то выбираем очередную входную точку и строим очередной многоугольник результата, как на шаге 4.

В литературе можно встретить следующее замечание: "Модификация этого алгоритма для определения части отсекаемого многоугольника, находящейся вне окна, заключается в следующем: а) исходная точка пересечения берется из списка выходных точек; б) движение по списку вершин окна выполняется в обратном порядке, т.е. так чтобы внутренняя часть окна была справа". Мы же можем такого замечания и не делать. Изложенный алгоритм, по сути, выполняет *операцию пересечения* двух многоугольников. Читатель может заметить, что можно просто поменять местами многоугольники, и результат будет тем же. В наших определениях для получения части  $M$ , лежащей вне окна, достаточно изменить направление обхода вершин  $W$ .

*Самостоятельно.* 1. В каких случаях списки точек пересечения пусты?

2. Как уточнить алгоритм на случаи, когда многоугольник и/или окно имеют многосвязную границу, т.е. имеют дырки?

Отметим, что было разработано множество алгоритмов отсечения в зависимости от конфигурации окна, например, отсечение по окну, граница которого состоит из отрезков и дуг окружностей [10].

Алгоритм Вейлера-Азертонса можно легко модифицировать для выполнения теоретико-множественных операций над многоугольниками.

## 5.5. Регуляризованные теоретико-множественные операции

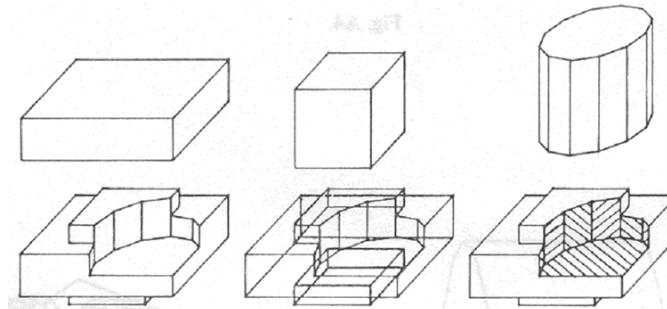


Рис. 5.10. Применение ТМО КОНТРАСТ

Из теории множеств нам известны теоретико-множественные операции (ТМО): объединение, пересечение, разность. Они достаточно просто и в конструктивной манере позволяют строить объекты: машиностроительные детали, обстановку трехмерной сцены и т.п. На

рис. 5.10 показан пример как из простых форм можно достаточно просто смоделировать деталь, применяя ТМО: из объединения двух брусков вычли цилиндр.

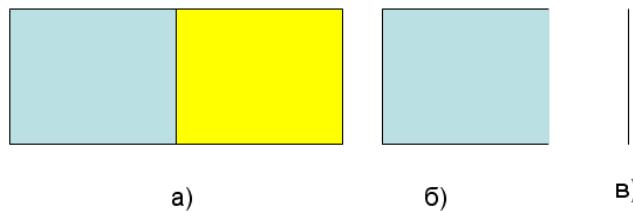


Рис. 5.11. Если многоугольники замкнутые множества

Читатель мог заметить, что пока мы не задумывались: какое множество представляет многоугольник: открытое, замкнутое, другое. В компьютерной графике и геометрическом моделировании применяются привычные для нас объекты, и, потому, на первый взгляд, удобно бы их рассматривать вместе с границей – замкнутые. В вычислительной математике области – это открытые множества. На рис. 5.11 представлен случай, когда многоугольники являются замкнутыми точечными множествами, и к ним применяются обычные ТМО:

- а) исходная сцена состоит из двух прямоугольников  $P_1$  и  $P_2$ , имеющих общую сторону;
- б) результат вычитания  $P_1 - P_2$ . Как видим, получился некоторый нереальный объект – это  $P_1$  без одной стороны, т.е. его граница не замкнута, такого в реальности не бывает;
- в) результат пересечения двух пристыкованных прямоугольников  $P_1$  и  $P_2$ . Получили отрезок, хотя в реальном мире два пристыкованных объекта не имеют общих точек.

Мы в определении, приведенном выше, упомянули, что многоугольник является частным случаем области и, таким образом, является открытым множеством. Читатель может самостоятельно убедиться, что теоретико-множественное объединение двух открытых многоугольников  $P_1$  и  $P_2$ , имеющих общий участок границы, даст несвязное множество.

Для того чтобы результат операции выглядел реалистично были предложены *регуляризованные теоретико-множественные операции* (РТМО) [12], обозначим их через  $\cup^*, \cap^*, \setminus^*$ , которые определяются следующим образом:

$$C = A \cup^* B = (\bar{A} \cup \bar{B})^\circ - \text{РТМ объединение многоугольников } A \text{ и } B. \text{ Обозначения:}$$

*черта сверху* – это замыкание множества;  $(\dots)^\circ$  – взятие внутренности.

$$C = A \cap^* B = (\bar{A} \cap \bar{B})^\circ - \text{РТМ пересечение многоугольников } A \text{ и } B.$$

$$C = A \setminus^* B = (\bar{A} \setminus \bar{B})^\circ - \text{РТМ разность многоугольников } A \text{ и } B.$$



Рис. 5.12. Результат вычитания – 4 многоугольника

Кстати, аналогичные РТМО можно определить и для других объектов – не только многоугольников. На рис. 5.12 представлена часто встречающаяся ситуация, когда в результате РТМО получается не один, а несколько многоугольников.

Обычно строится алгебра над объектами следующего вида:

- Конечный набор попарно непересекающихся многоугольников – это объект.

- Пустое множество  $\emptyset$  тоже объект.
- Вся плоскость  $R^2$  тоже объект.

Можно показать, что данное множество объектов замкнуто относительно введенных РТМО.

Читатель может сделать замечание, что мы работаем с открытыми точечными множествами, судя по формулам. Да, это так. Просто во всех случаях использования наших объектов, например для растеризации, мы берем их замыкание. Если же мы будем рассматривать их замкнутыми, то возникают проблемы, как на примере рис. 5.13, где две части объекта имеют общую вершину, т.е. возникает связность.

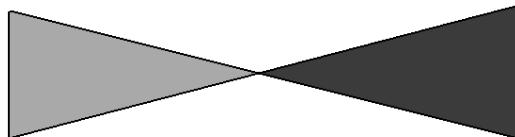


Рис. 5.13. Общая вершина

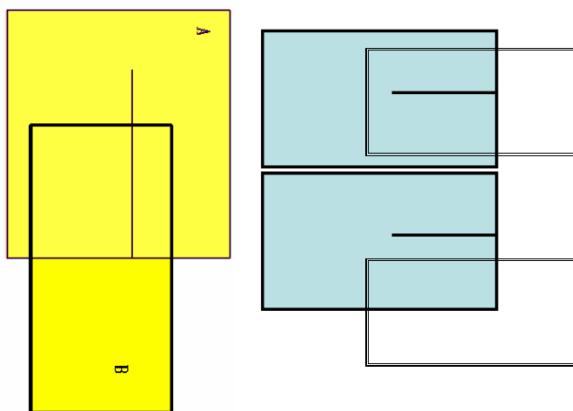


Рис. 5.14. ТМО над областями с разрезами

В реальных программных продуктах возможно применение *нерегуляризованных* ТМО к объектам более широкого класса, если не возникает двусмысленности при трактовке результата операции. Но, как говорится, на страх и риск пользователя. Например, мы хотим работать с областями, имеющими разрезы. На рис. 5.14 рассмотрены различные случаи пересечения двух прямоугольников, один из которых с разрезом. Два случая справа дают вполне понятные результаты: прямоугольник с разрезом, прямоугольник. А вот случай слева дает два прямоугольника с общей стороной. При использовании регуляризованных ТМО этого не возникает, т.к. разрез исчезнет на первом шаге операции, когда выполняется замыкание объекта.

Реализация алгоритма РТМО может быть выполнена аналогично алгоритму Вейлера-Азертона и заключается в следующих шагах:

1. Разбиение границы одного операнда границей другого операнда.
2. Получение непересекающихся частей  $A \setminus^* B, A \cap^* B, B \setminus^* A$ .
3. Компоновка результата в случае операции  $A \cup^* B$ .

После первого шага выполнение операций сводится к сортировкам списков вершин и точек пересечения примерно так, как это делалось в алгоритме Вейлера-Азертона.

## 5.6. Растеризация многоугольников

После отрезка многоугольник является самой популярной фигурой. Как правило, имеются в виду *ограниченные* многоугольники, возможно с дырками, т.е. их граница – это несколько

контуров. Случай граничного представления многоугольника на растре ничем не отличается от случая растеризации отрезков и был рассмотрен ранее. Задача заключается в закраске (часто говорят "заливке") всех пикселей, принадлежащих многоугольнику вместе с границей, вершины которого имеют целочисленные координаты. В данном курсе мы не будем подробно останавливаться на всем многообразии алгоритмов преобразования многоугольников в пиксельные области, с которыми можно ознакомиться, например, в книгах [9, 11]. На рис. 5.15 представлен многоугольник  $M$ , граница которого состоит из одного внешнего контура (точки 1 – 7) и одного внутреннего – точки 8 – 10. Простейший алгоритм заливки многоугольника можно описать следующим образом:

1. Растеризуем все контуры границы по алгоритму Брезенхэма. На рис. 5.15 мелкие кружки представляют центры пикселей, которые аппроксимируют границу. Получили 8-связную пиксельную область.
2. Применяем любой алгоритм заливки пиксельных областей. Затравка здесь нам не нужна, т.к. мы договорились, что многоугольник ограниченный.

*Сканирующая линия* – строка пикселей растра, горизонтальная линия на растре или на растровой плоскости. При описании алгоритмов сканирующие линии или строки часто нумеруются сверху вниз, по аналогии с движением сканирующего луча дисплея. В данном тексте линии нумеруются снизу вверх, и  $y$ -координата пикселей также растет вверх, см. рис. 5.15.

*Активной линией растра* для многоугольника называется строка пикселей, имеющая с многоугольником непустое пересечение. Для  $M$  на рис. 5.15 показано 23 активных линий. Линии 4 – 19 – это активные линии для ребра [6, 7].

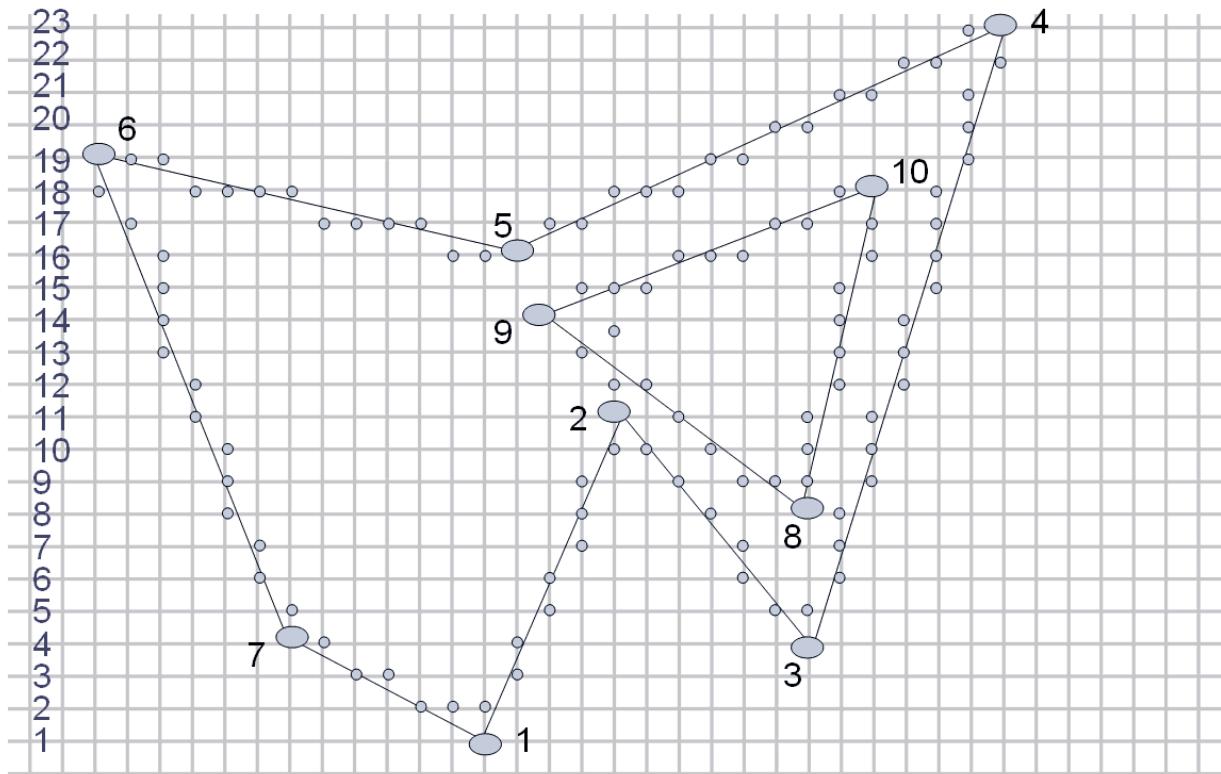


Рис. 5.15. Представление границы многоугольника на растре

Введем *порядок сканирования* " $\triangleleft$ " на множестве вершин  $M$ :  $V_1 = (x_1, y_1), \dots, V_n = (x_n, y_n)$ . Будем говорить, что  $V_i \triangleleft V_j$ , если  $y_i < y_j$ , а при равенстве  $y_i = y_j$ , если  $x_i < x_j$ .

На основе понятия характеристической функции, введенной ранее, определим следующие правила или факты:

- Бесконечная прямая пересекает замкнутую ограниченную область четное число раз кроме случаев, когда прямая проходит по сторонам или по вершинам.
- Мы можем заполнять многоугольник, перебирая последовательные сканирующие линии. При этом заполняются пиксели, расположенные между точками пересечения очередной линии со сторонами.
- Не нужно обрабатывать горизонтальные стороны, т.к. эти пиксели находятся между двумя вершинами и будут закрашены.
- Если сканирующая линия попадает на вершину, то:
  - 1) записываем два пересечения, если эта вершина является локальным максимумом или локальным минимумом. Это вершины 1, 6, 5, 4, 3, 2, 8, 10.
  - 2) в противном случае записываем одно пересечение – вершины 7 и 9.

### 5.6.1. Простейший алгоритм

1. Заводим список, в котором будем хранить граничные пиксели. Выполняем растеризацию сторон и определяем все пиксели, принадлежащие границе. Все эти пиксели записываем в список.
2. Согласно порядку сканирования " $\triangleleft$ " сортируем все граничные пиксели в списке.
3. Выбираем попарно точки  $(x_1, y_1)$  и  $(x_2, y_2)$  из списка и красим каждый пиксель  $(x, y)$ , удовлетворяющий условию  $x_1 \leq x \leq x_2$ . Согласно нашему предположению о четности должно выполняться  $y_1 = y = y_2$ , т.е. каждая пара принадлежит одной линии сканирования.

*Упражнение.* Нами рассмотрен алгоритм заливки внутренности многоугольника вместе с пикселями границы. Как надо его скорректировать, чтобы заливались только внутренние пиксели?

### 5.6.2. Алгоритм ключевых ребер

В реальных программах полный список граничных пикселей, как правило, не формируется, поскольку растр лазерного принтера достаточно мелкий, а общее количество точек пересечения (граничных точек) может быть огромным. Тогда операция сортировки будет занимать значительное время. Пуще обрабатывать сканирующие линии по одной, создавая список для текущей линии. Более того, не все пиксели надо сохранять, мы можем рассчитывать их на лету – по мере необходимости. Действительно, зачем делать предварительную растеризацию сторон и создавать огромный список? Рассмотрим более практичный алгоритм. Назовем *ключевыми* пикселями пиксели, содержащие вершины многоугольника.

Для сканирующей линии у о каждой стороне многоугольника достаточно хранить тройку величин  $\langle x, \delta y, \delta x \rangle$ . Рассмотрим негоризонтальную сторону с вершинами  $(x_1, y_1)$  и  $(x_2, y_2)$ , и пусть  $(x_1, y_1) \triangleleft (x_2, y_2)$  для определенности.  $\delta y = y_2 - y_1 + 1$  – число активных сканирующих линий, пересекаемых этой стороной. Если начинать с нижнего пикселя стороны  $(x_1, y_1)$ , то при изменении  $y$  на 1 (переход на следующую сканирующую строку),  $x$  изменится на  $1/m$ , где  $m$  – наклон стороны.

*Предварительная подготовка.* Для каждой негоризонтальной стороны  $[(x_1, y_1), (x_2, y_2)]$ ,  $(x_1, y_1) \triangleleft (x_2, y_2)$ , определим значения  $\delta y, \delta x, x$ :

- $\delta y = y_2 - y_1 + 1$  – число активных линий раstra данной стороны;
- $\delta x = \frac{x_2 - x_1}{y_2 - y_1}$  – приращение координаты  $x$  при переходе к следующей сканирующей линии, может быть отрицательным;
- $x = x_1$  – координата  $x$  граничного пикселя на текущей сканирующей линии, сначала на первой;
- $y = y_1$  – текущая сканирующая линия, сначала нижняя.

*Корректировка.* Если  $(x_1, y_1)$  не является локальным минимумом, то производим корректировку:  $y = y_1 + 1$  и  $x = x_1 + \delta x$ . Действительно для вершины 7 сканирующая линия  $y = y_7$  будет обработана во время работы с отрезком  $[1, 7]$ , и мы сразу настраиваем данные на следующую сканирующую линию.

Все эти данные отсортируем по возрастанию согласно порядку " $\triangleleft$ " и сформируем из них список ключевых точек  $LKP$ . Тогда алгоритм записывается следующим образом.

*Начало.* Введем еще один список – список активных сторон  $LAS$ . Начинаем с нижней сканирующей линии раstra. Выбираем из  $LKP$  все записи запомненных данных для нижней строки раstra  $y$  и помещаем их в  $LAS$ .

1. Закрашиваем все точки (пиксели) сканирующей линии между значениями  $x$  каждой пары.
2. Корректируем  $\delta y = \delta y - 1, x = x + \delta x$ .
3. Если  $\delta y$  стало равным 0, убираем соответствующую запись данных из списка активных сторон  $LAS$ .
4. Увеличиваем номер сканирующей линии  $y$  на 1.
5. Просматриваем начало  $LKP$  и если для строки  $y$  в нем имеются пары данных, то мы их добавляем в  $LAS$  и сортируем его согласно порядку (в примере рис. 5.15 такая ситуация встретится нам на линии 4 из-за 3-й вершины, на линии 5 из-за 7-й, на линии 8 из-за 8-й, на линии 15 из-за 9-й, и т.д.).
6. Повторяем шаги 1 – 5 до тех пор, пока список  $LAS$  не станет пуст.

*Вопрос.* Как можно поступать в случаях, когда многоугольник выходит за границы раstra?

## 6. Работа с цифровыми изображениями

Данная тема включена в курс для общего ознакомления учащихся с задачами обработки изображений. Изложение теоретических основ не входит в цели данного курса бакалаврской подготовки, поэтому после ряда попыток авторы остановились на форме подачи материала, базирующегося на примерах, которые достаточно часто встречаются в практической деятельности информационного технолога. Те же из студентов, которые продолжат обучение в магистратуре по специальности "Системы мультимедиа и компьютерная графика", восполнят недостающий теоретический базис. Отмечу, что примерно такой же подход к изложению задач обработки изображений применяется в большинстве вводных курсов компьютерной графики.

### 6.1. Что такое пиксель?

*Пиксель – это квадрат.*

Для понимания алгоритмов предыдущего материала было достаточно идентифицировать пиксель его центральной точкой. Здесь мы переходим к изображениям, и, рассматривая пиксель как часть этого изображения, мы должны вспомнить про его площадь. Эксплуатируется следующая основная идея. Пиксель имеет ненулевую площадь на экране, значит, он должен быть использован для визуального представления области мирового пространства с ненулевой проекционной площадью – областью, которая отображается в этот пиксель.



Рис. 6.1. Пиксель – это часть отображаемого мира **ЗАМЕНИТЬ ВИН**

Поскольку пиксель дальше уже не делится, он содержит всего один цвет эквивалентный по энергии и восприятию (усредненный) световому потоку, приходящему в телесном угле, стянутом границами этого пикселя. Отметим, что человеческий глаз также имеет конечное разрешение – "пиксель глаза" порядка 1 минуты по углу.

Стоит заметить, впрочем, что попытки делить пиксель предпринимаются. Наиболее известная — это субпиксельное отображение шрифтов в современных графических системах (к примеру, технология ClearType в системе Microsoft Windows). Здесь используется физическая природа современных LCD-дисплеев, в которых квадратный пиксель состоит из трёх вертикальных полосок-субпикселей: красной, зелёной и синей. Смесь этих цветов даёт белый, однако никто не мешает смешивать, к примеру, зелёный и синий субпиксели одного пикселя с красным субпикселям его правого соседа. Применяя этот подход, можно увеличить эффективное разрешение по горизонтали втрое, в результате чего текст на экране будет выглядеть более сглаженным. На рис. 6.2. показано, как выглядит слово «пиксель», растеризованное с помощью ClearType, при 10-кратном увеличении.



Рис. 6.2. Субпиксельная растеризация шрифта

На самом деле ряд авторов отстаивает мнение, что пиксель не квадрат, а выборка. Но в данном изложении модель пикселя-квадрата вполне адекватна. Хотя авторы и взяли в

качестве основы материала данной главы курс COS 426, Spring 2005 проф. Adam Finkelstein из Princeton University (<http://www.cs.princeton.edu/courses/archive/spr05/cos426/>).

## 6.2. Задача аппроксимации полутонон

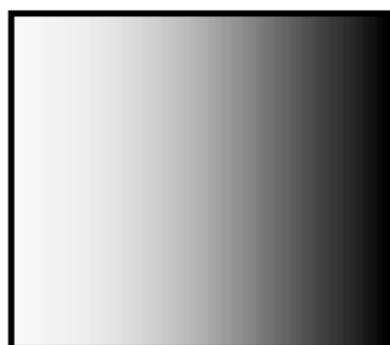
В данной части курса считается, что все цветные изображения представлены в цветовом пространстве RGB. Как правило, если не оговаривается специально, все три канала рассматриваются и обрабатываются отдельно и независимо. Таким образом, мы будем рассматривать монохромные или полуточновые изображения, а в необходимых местах вспоминать о цветах.

*Палитра* – множество оттенков или полутонон, обеспечиваемых устройством для использования в изображении.

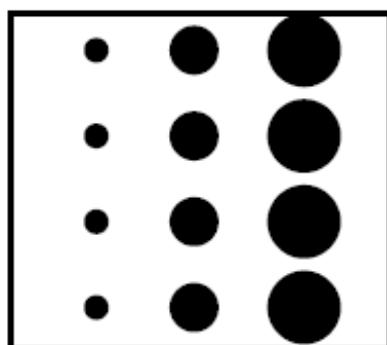
Крайние полутона – это *черный* (значение 0) и *белый* (значение 1). Остальные полутона с некоторой степенью равномерности лежат в диапазоне от 0 до 1. Часто применяемые числовые значения полутонон 0, 1, 2, ..., 255 являются, по сути, номерами (именами), но в ряде случаев эти номера могут успешно нести нагрузку самих значений, а диапазон 0–255 отображается в диапазон 0–1.

Аппроксимация полутонон (или часто используется термин *дизеринг* – dithering) требуется при необходимости представить на устройстве с одной палитрой (одним количеством воспроизводимых полутонон) изображение, рассчитанное для другой палитры.

Исторически эта задача решалась очень давно: газетная печать. Перевод черно-белого полуточнового изображения на бумагу достигается за счет варьирования пространственной (площадной) "яркости" (черноты). На рис. 6.3 представлен полуточновой рисунок и его точечное воспроизведение. Здесь и далее:  $I(x, y)$  – значение интенсивности исходного изображения в пикселе  $(x, y)$ , а  $P(x, y)$  – значение в выходном изображении.



$I(x, y)$



$P(x, y)$

Исходное изображение

Представление в газете

Рис. 6.3. Передача "яркости" в газете

Для конца прошлого века были характерны следующие показатели:

- Газета: 20-30 точек/дюйм.
- Журнал: порядка 60 точек/дюйм

Но точки могут быть разного диаметра. Другими словами, задача решается за счет увеличения пространственного разрешения. На рис. 6.4. приведен реальный пример.

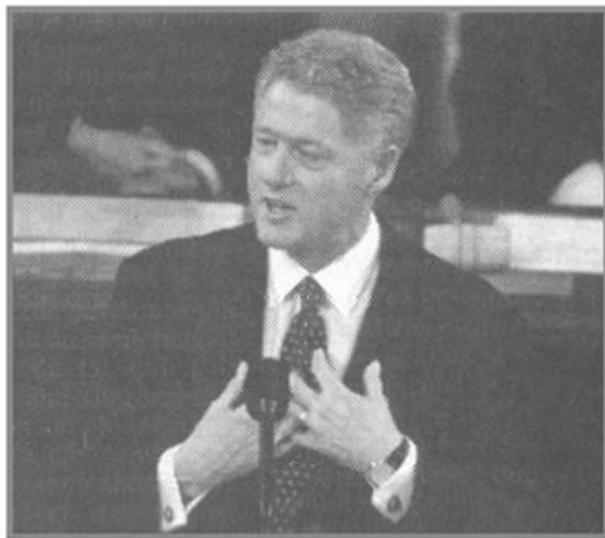


Рис. 6.4. Реальный пример газетной печати

Тагир, а мы не сможем что-то Российское здесь вставить на рис. Вместо Клинтона?

Для чего нам нужен дизеринг?

- Чтобы печатать на принтерах. По каждому цвету принтер дает только 0 и 1, например, для жёлтого: есть или нет. Только новейшие технологии печати (в частности, термическая сублимационная печать) позволяют выводить настоящие полутоновые изображения на бумагу. Обычные лазерные и струйные принтеры такой возможностью не обладают.
- Чтобы преодолеть ограничения, накладываемые устройством вывода (дисплей, видеокарта). На современных дисплеях это не особенно актуально, однако, к примеру, заставка Windows XP выводится с применением дизеринга, так как в это время драйвер видеокарты ещё не инициализирован и возможности для вывода ограничены.
- Из-за ограничений, накладываемых применяемым форматом файла (в частности, GIF).
- Для уменьшения памяти, используемой для хранения изображения.

На рис. 6.5. приводится очень показательный пример.

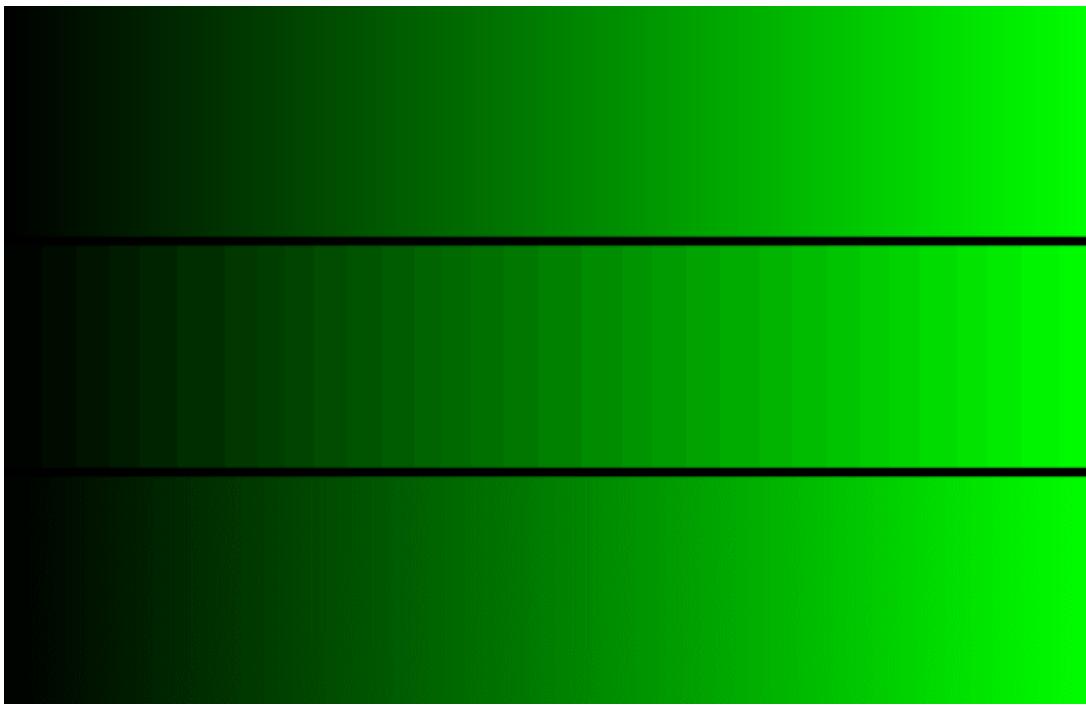


Рис. 6.5. Верхняя полоса – исходное изображение в палитре из 256 полутона. Средняя полоса – это же изображение в палитре из 32 полутона. Низ – палитра 32 полутона, но применен дизеринг

### ***6.3. Аппроксимации полутонов с увеличением пространственного разрешения***

Такой подход мы уже рассмотрели при газетной печати, рис. 6.2. Задачу можно охарактеризовать следующим образом: *каждый пиксель исходного изображения можно представлять, используя несколько пикселей носителя выходного изображения*. Основной принцип: "картины смотрят, а не нюхают".

Как создаются палитры выходного устройства? На первых печатающих устройствах ЭВМ устройства позволяли печатать на литерных площадках одного размера только алфавитно-цифровые и специальные знаки. Конструкторы палитр подбирали наборы знаков, обеспечивающих монотонную и почти равномерную по яркости последовательность полутонов.

После появления знако-синтезирующих принтеров стали рассматриваться и другие подходы. Раз можно увеличить пространственное разрешение, а у лазерного принтера пиксель намного меньше, чем у экрана, то простейший способ заключается в следующем. Поле вывода разбивалось на одинаковые квадраты, в каждый из которых входило  $k \times k$  пикселей. Таким образом, такой квадрат мог представлять палитру из  $k \times k + 1$  полутона или "яркостей" – по количеству черных пикселей. Глаз осуществляет усреднение по пространству, т.е. надо создать шаблоны такие, что они дают требуемые усредненные яркости.

Правило соответствия полутонов зачерненным пикселям задается матрицей дизеринга. Был выработан ряд правил, одно из которых требует, чтобы пиксель, появившись в одном полутоне, должен присутствовать и на более высоких номерах полутонов. Тогда и записи стали компактны. Например, матрица дизеринга

$$\begin{matrix} 6 & 8 & 4 \\ 1 & 0 & 3 \\ 5 & 2 & 7 \end{matrix}$$

задает палитру из  $10 (=3 \times 3 + 1)$  полутонов. Цифры в матрице соответствуют номерам полутонов, после которых данный пиксель зачерняется. На рис. 6.6 простейший пример палитры.

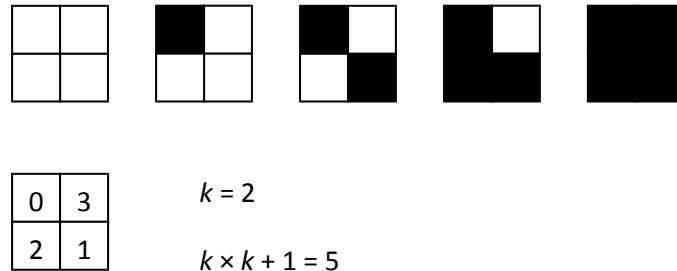


Рис. 6.6. Палитра на 5 полутонов

Есть еще ряд правил, например, рассмотрим 4-й полутон (рис. 6.7, слева), построенный по приведенной выше матрице.

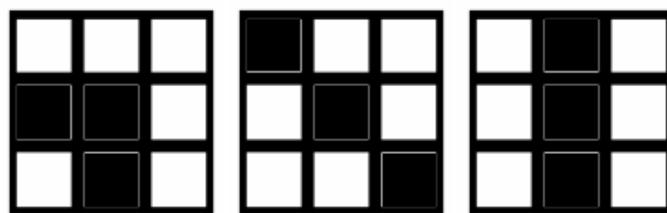


Рис. 6.7. Вид 4-го полутона (слева) и неудачные конструкции 4-го полутона

На рис. 6.7 (средний и справа) показаны неудачные конструкции этого полутона, поскольку области постоянной яркости исходного изображения будут выглядеть на выходном изображении заштрихованными.

## 6.4. Аппроксимации полутонов без увеличения пространственного разрешения

Задача формулируется следующим образом: пересчитать изображение с  $N$  интенсивностями без изменения разрешения в палитру, состоящую из  $K$  интенсивностей.

Мы рассмотрим алгоритмы для следующих случаев:

- $K = 2$  — возможны только два цвета (к примеру, чёрный и белый). Будем обозначать их 0 и 1.
- Произвольное  $K$ .

Отметим, что ряд алгоритмов рассматривает каждый пиксель по-отдельности, другие работают с учетом значений соседних пикселей.

### 6.4.1. Попиксельное равномерное квантование

Итак, на вход поступает значение  $I(x, y)$ , мы должны получить выходное значение  $P(x, y)$ , которое является ближайшим к  $I(x, y)$  из всех значений выходной палитры. Это метод *равномерного квантования*, см. рис. 6.8.

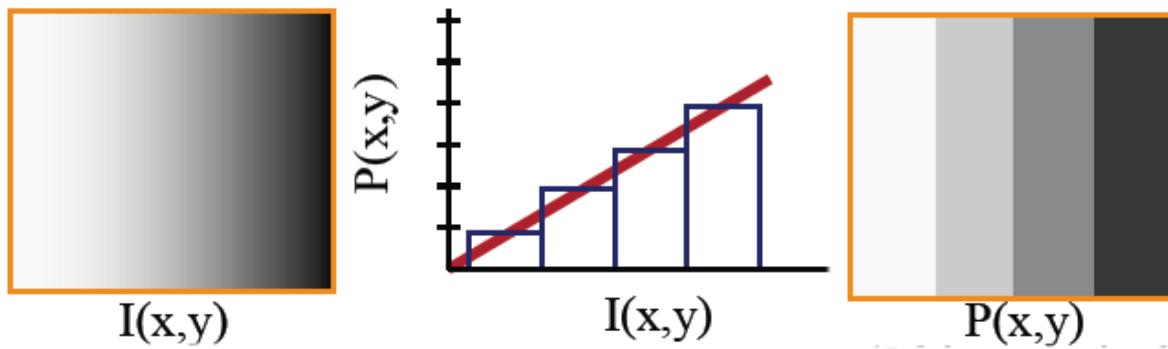


Рис. 6.8. Равномерное квантование

Какие мы видим артефакты на выходном изображении? Четко выраженные полосы, которые возникли из-за *ограниченного разрешения по интенсивности* – бедности выходной палитры. Это называется оконтурированием (banding), возникающим из-за перехода на бедную палитру. На рис. 6.9 выходная палитра из 4-х значений, т.е. под значение пикселя выделено 2 бита. На рис. 6.10 (знакомьтесь с проф. Финкельштейном Деб: может заменим его надр. Изображение? А?) приведены результаты равномерного квантования при различных палитрах.



Рис. 6.9. Слева направо палитры: 8 битов, 4 бита, 2 бита, 1 бит. Заметно оконтуривание **НА ЛЕНУ**

Как уменьшить погрешности равномерного квантования? Классический метод передачи полутона (halftoning) был рассмотрен нами выше – газетная печать – рис. 6.4. Но у нас нет возможности варьировать пространственное разрешение, следовательно, необходимо искать другие подходы.

#### 6.4.2. Попиксельный дизеринг

Смысл дизеринга заключается в том, чтобы эксплуатировать интегрирование по телесному углу – свойство, присущее нашему глазу. Возникшую ошибку мы должны как-то восполнить в выходном изображении и, таким образом, отображать больший диапазон воспринимаемых интенсивностей.

Черно-белый дисплей имеет палитру из 2-х оттенков: 0 и 1. Предположим, что мы собираемся переводить изображение, представленное слева на рис. 6.10.



Рис. 6.10. Слева: исходное изображение, палитра 0-255, 8 бит. Середина: равномерное квантование, палитра: 0 и 1. Справа: с применением шума **СДЕЛАТЬ РАНД ДЛЯ ЛЕНЫ**  
Ошибка, возникшая из-за квантования, – это по сути дела некоторый шум – *Noise*, а что если мы к исходному изображению будем просто добавлять шум перед квантованием, как это показано на рис. 6.11. В этом случае расчетная формула выглядит так:

$$P(x, y) = \text{Round}(I(x, y) + \text{Noise}(x, y)).$$

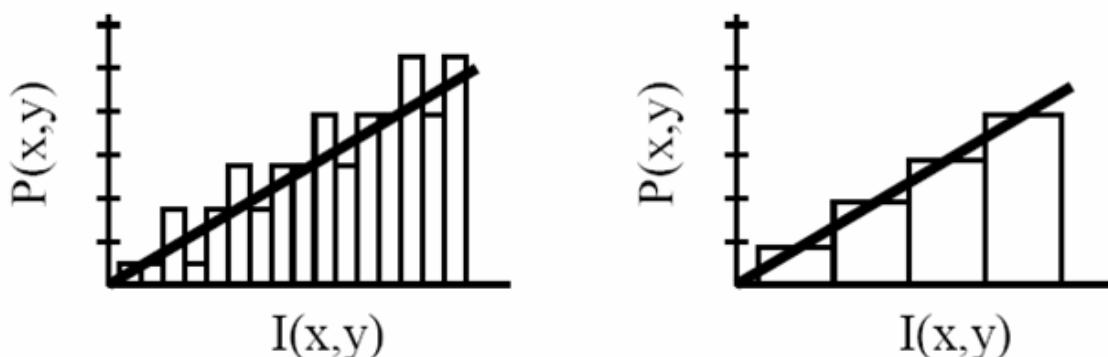


Рис. 6.11. Слева: квантование с шумом, справа: равномерное квантование

Результат показан на рис. 6.10 справа, и он оказался еще более плохим. Но, с другой стороны, на нем мы обнаруживаем дополнительные детали исходного изображения в отличие от равномерного квантования (рис. 6.10, середина).

Хочется отметить, что аналогичный метод применяется не только в обработке изображений, но и в обработке сигналов вообще. К примеру, при уменьшении громкости звукового сигнала вдвое у вас так же вдвое снижается его разрешение. Если вы просто поделите значения амплитуды сигнала на два и округлите, в спектре сигнала появится нежелательная высокочастотная компонента, которая будет восприниматься как писк. Добавление случайного шума нивелирует этот эффект.

Рассмотрим другой метод, широко применяемый на практике, – *метод упорядоченного возмущения* (ordered dither), который основывается на матрицах порогов. Байер предложил матрицы дизеринга, определяющиеся на основе рекуррентного соотношения.

- $D_2 = \begin{pmatrix} 3 & 1 \\ 0 & 2 \end{pmatrix}$
- $D_{2n} = \begin{bmatrix} 4D_n + 3U_n & 4D_n + U_n \\ 4D_n & 4D_n + 2U_n \end{bmatrix}$ , где  $U_n$  – это матрица размером  $n \times n$ , состоящая из всех единиц.

Пример:

$$D_4 = \begin{pmatrix} 15 & 7 & 13 & 5 \\ 3 & 11 & 1 & 9 \\ 12 & 4 & 14 & 6 \\ 0 & 8 & 2 & 10 \end{pmatrix}.$$

Матрицы, построенные таким образом, хороши тем, что с ними не возникает таких полос, как на рис. 6.7 (средний и справа). Необходимый размер матрицы дизеринга выбирается исходя из размера входной палитры. Требуется, чтобы число элементов матрицы при минимальном  $n$  было не меньше, чем число полутоонов в исходной палитре. Принимая во внимание, что входная палитра обычно характеризуется числом битов для нумерации полутоонов, то выбирается такая матрица дизеринга, которая состоит из всех полутоонов исходной палитры. Для исходной палитры 0–255 требуется матрица  $D_{16}$ . Сам алгоритм упорядоченного дизеринга в простейшем случае записывается следующим образом:

1. Определили  $n$
2. Сначала вычисляется блок размером  $n \times n$  в исходном изображении (как на рис. 6.12), в который попадает пиксель  $(x, y)$  и индексы пикселя в этом блоке:  $i = x(\text{mod}_n)$ ,  $j = (y \text{ mod}_n)$ . Деб: в конце надо привести все матформулы к одному стилю и размеру. Это делается на раз, поэтому счас не берем в голову.
3. Затем вычисляется  $P(x, y) = \begin{cases} 1, & I(x, y) > D_n(i, j) \\ 0, & I(x, y) \leq D_n(i, j) \end{cases}$ . Если число уровней яркости изображения не соответствует в точности максимальному значению матрицы, то перед этим шагом  $I(x, y)$  необходимо перенормировать.

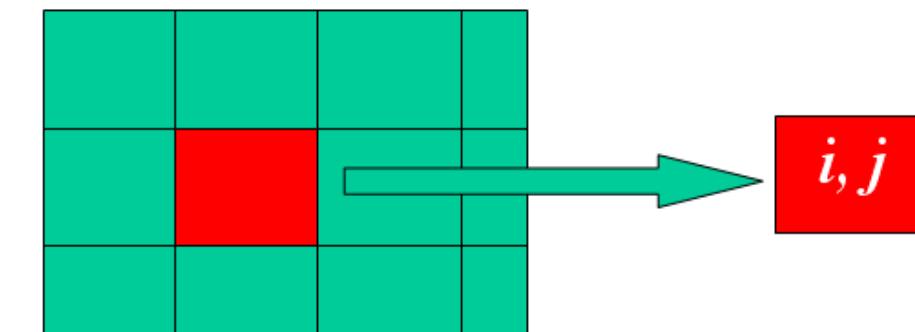


Рис. 6.12. Алгоритм ordered dither

Конечно, применение данного метода дает на выходном изображении определенные муары и повторяющиеся текстуры, но тем не менее, начинают проявляться полутона, по сравнению с равномерным квантованием.



Рис. 6.13. Слева – оригинал; середина – квантование с шумом; результат упорядоченного дизеринга – справа

Отметим, что такой дизеринг с матрицей  $2 \times 2$  применялся уже в ранних ТВ приемниках. Конечно, имеются и более сложные варианты попиксельного дизеринга.

Но ресурсы попиксельных алгоритмов сильно ограничены! Ведь эксплуатируя интегрирующие свойства человеческого глаза, мы можем задействовать значения близких пикселей.

### 6.4.3. Диффузия ошибки

*Постановка задачи:* пересчитать изображение с палитрой из  $N$  интенсивностей без изменения разрешения в палитру, состоящую из  $K$  интенсивностей.

Введем величины:

- $di = \frac{1}{N-1}$  – квант исходной палитры;
- $do = \frac{1}{K-1}$  – квант выходной палитры.

Входная палитра представлена значениями:  $\{i \times di\}, i = 0, \dots, N-1$ .

Выходная палитра представлена значениями:  $\{j \times do\}, j = 0, \dots, K-1$ .



Рис. 6.14. Вверху исходное изображение, внизу – результат дизеринга фрагмента **ТАГИР**

Идея алгоритма состоит в том, чтобы сохранять энергетический баланс всего изображения: если мы сделали пиксель темнее, то должны компенсировать это, добавив света в окружающие пиксели, и наоборот. В популярном алгоритме Флойда-Стейнберга разность

яркостей или ошибка распределяется по четырём соседним пикселям. Ошибка делится неравными долями, чтобы снизить шансы возникновения муаров, причём 4-связные пиксели получают больше, чем 8-связные (рис. 6.15).

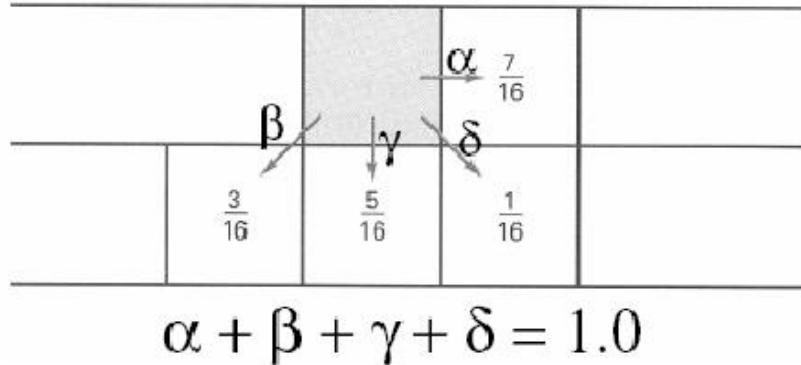


Рис. 6.15. Распределение ошибки по соседям (Перерисовать НАДО)

Основываясь на таком распределении ошибки, запишем алгоритм Флойда-Стейнберга:

```
for (x = 0; x < width; x++) // цикл по столбцам
    for (y = 0; y < height; y++) // цикл по строкам
    {
        P(x, y) = ближайшее к I(x, y) значение в выходной палитре;
        error = I(x, y) - P(x, y);
        I(x+1, y) = I(x+1, y) + 7/16 * error;
        I(x-1, y+1) = I(x-1, y+1) + 3/16 * error;
        I(x, y+1) = I(x, y+1) + 5/16 * error;
        I(x+1, y+1) = I(x, y+1) + 1/16 * error;
    }
```

Таким образом, ошибка может быть со знаком:

- Положительная. Значит, не вся энергия пикселя передана в выходное изображение. Ее по частям добавляем к еще не обработанным пикселям-соседям.
- Отрицательная. В выходное изображение передана лишняя энергия и равное ей количество отнимается от еще не обработанных пикселей-соседей.

Вопросы к читателю:

- Почему ошибка распределяется только в правый и три нижних пикселя, но не распределяется влево и вверх?
- Какой минимальный объём памяти требуется для работы алгоритма, если предположить, что алгоритм работает в потоковом режиме (к примеру, последовательно вычитывает пиксели изображения из файла и записывает результат в другой файл)?

Всегда задается вопрос, что делать с крайними пикселями. Есть масса решений, одно из которых: если доля ошибки добавляется (отнимается) к несуществующему пикслю, то эту долю просто игнорировать. Немного приложений можно придумать, где важна граничная полоса шириной 1-2 пикселя. К примеру, ваше приложение ищет на изображении знакомое лицо, какое значение имеет граничный бордюр шириной в пару пикселей?

На рис. 6.16 для сравнения приведено несколько способов дизеринга при K=2, N=256.

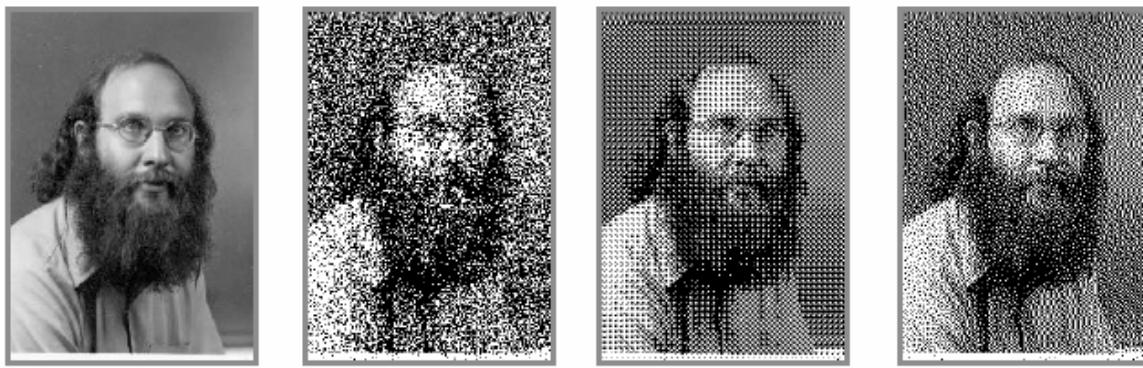


Рис. 6.16. Слева направо: оригинал, квантование с добавлением шума, ordered dither, Флойд-Стейнберг

Было разработано достаточно много разновидностей распределения ошибки между соседями, некоторые из них показаны на рис. 6.17.

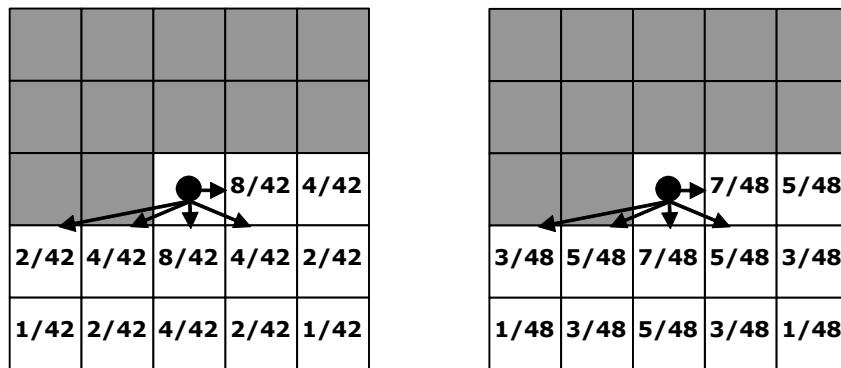


Рис. 6.7. Другие способы распределения ошибки

И, наконец, на сводном рис. 6.18 приведены результаты сразу нескольких способов дизеринга на другом тестовом изображении.



Рис. 6.18. Известное тестовое изображение Lena

## 6.5. Гамма-коррекция

Про многие явления (вкус, запах и т.д.) можно говорить, что они имеют большую или меньшую интенсивность. И эту интенсивность можно измерить в вполне конкретных физических величинах. Человек воспринимает эти же явления по-своему – в несколько другом масштабе – см. табл. 6.1. Закон Стивенса гласит, что зависимость субъективного ощущения  $S$  от действительной интенсивности  $I$  выглядит как:

$$S = I^p.$$

Таблица 6.1. Показатели восприятия некоторых явлений

| Исследуемое ощущение | Экспериментально полученный показатель $p$ |
|----------------------|--------------------------------------------|
| Яркость              | 0.33                                       |
| Запах                | 0.55                                       |
| Громкость            | 0.60                                       |
| Вкус                 | 0.80                                       |
| Длина                | 1.00                                       |
| Тяжесть              | 1.45                                       |

Разные авторы предлагали различные аналитические выражения зависимости яркости от интенсивности. Дополнительные факты об интересующей нас яркости:

- Стивенс –  $S = \sqrt[3]{I}$ . И, правда, поменяв в комнате лампочку с 50 до 100 ватт, мы не ощущаем удвоения освещенности в комнате.
- Фешнер –  $S = k \log I$ . Он предлагает рассматривать логарифмическую зависимость яркости от интенсивности.
- Вебер –  $JND = \frac{\Delta I}{I} \approx 0.01$ . Средний порог чувствительности к изменениям интенсивности – около 1%.

### 6.5.1. Гамма монитора

Итак, мы продолжаем рассматривать ахроматический цвет (черно-белый, серый). Яркость – это воспринимаемая характеристика, а интенсивность – это энергетическая характеристика. Иногда их следует различать. Измеряем их от 0 (черный) до 1 (белый). Для телевизионных трубок (и мониторов) зависимость интенсивности  $I$  от подаваемого напряжения  $V$  описывается формулой:

$$I = g \cdot (V - V_b)^\gamma,$$

где  $V_b$  – это ручка яркости на нашем ТВ,  $g$  – ручка контраста.  $\gamma$  – некоторая константа, характеризующая люминофор. Именно эта буква ГАММА и дала название специфическим проблемам. Для большинства мониторов гамма лежит в пределах от 2.2 до 2.5. Рассмотрим типичный современный монитор. Он позволяет управлять интенсивностью подсветки пикселя в диапазоне от 0 до 255. Приняв постулат, что человек воспринимает повышение яркости относительно, т.е. логарифмически, составим следующие соотношения:

- Диапазон интенсивностей – от 0.0 до 1.0.
- $I_1$  – минимальная интенсивность, она соответствует значению интенсивности 1.

- $I_2 = r \cdot I_1$ , соответствует значению 2.  $r$  определяет постоянное соотношение между соседними значениями интенсивности.
- $I_3 = r \cdot I_2$  для значения 3.
- ...
- $I_{255} = r \cdot I_{254} = 1.0$  для значения 255.
- Таким образом, получаем  $r = \left(\frac{1}{I_1}\right)^{\frac{1}{254}}$ . Это около 1.01, что хорошо согласуется с выводами Фешнера (см. выше). Если взять  $r = 1.01$ , то получим, что максимально человек может различить 463 уровня яркости. Отсюда вполне понятно, что отводить больше, чем 8 разрядов на цвет не очень рационально, поскольку среднестатистический человек этого все равно не ощутит.
- Для  $j$ -ой интенсивности  $I_j = r^{j-1} \cdot I_1$ .
- Зависимость яркости люминофора от напряжения –  $I = c \cdot V^\gamma$ , либо от числа  $N$  электронов в пучке –  $I = k \cdot N^\gamma$  ( $c$  и  $k$  – некоторые константы).
- $V = \left(\frac{I}{c}\right)^{\frac{1}{\gamma}}$ .
- $V_j = \left(\frac{I_j}{c}\right)^{\frac{1}{\gamma}}$  – показывает необходимое значение подаваемого напряжения, чтобы выдать в пикселе  $j$ -ю интенсивность.

Таким образом, мы рассчитали напряжения, чтобы работать с линейной палитрой в области восприятия. Но реальные люминофоры работают не точно по рассмотренному закону. В связи с этим приходится осуществлять корректировку таблицы напряжений для каждого значения интенсивности  $n$  от 0 до 255. Такая корректировка формы кривой возможна на программном уровне. Влияние значения гаммы на восприятие изображения показано на рис. 6.19.

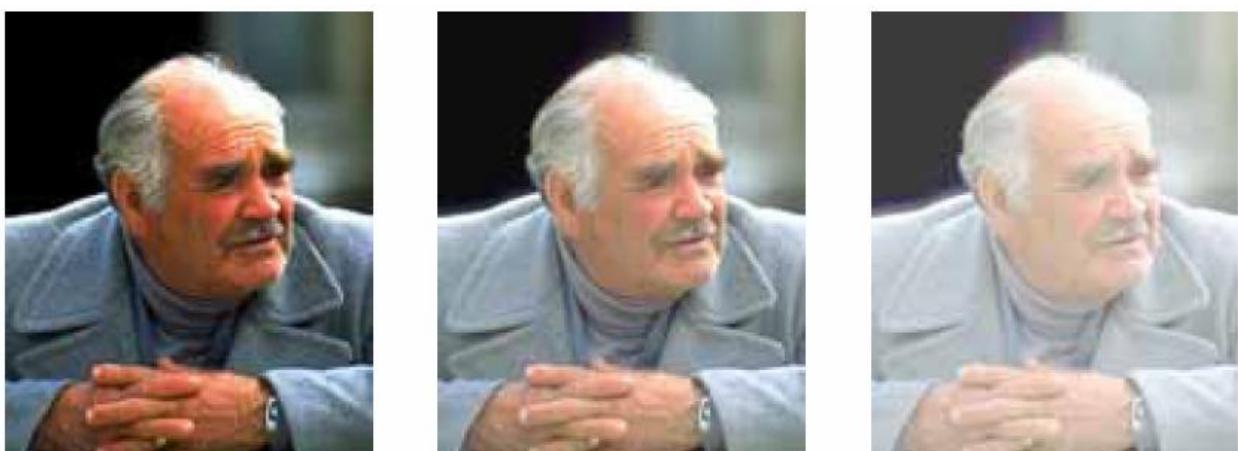


Рис. 6.19. Слева: корректировка под гамму меньшую, чем гамма дисплея; середина: скорректировано под правильную гамму; справа: грамма больше гаммы дисплея

**ПЕРЕРИСОВАТЬ ПОД ЛЕНУ**

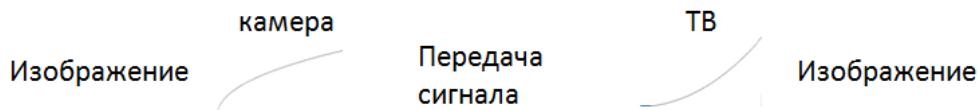


Рис. 6.20. Слева – кривая с гаммой камеры (обычно 0.45),  
справа – кривая с гаммой дисплея 2.5

Деб: также надо сбацать графики почетче-жирнее

Если рассмотреть процесс получения и отображения изображений (рис. 6.20), то понятно, почему на ТВ мы наблюдаем нормальные изображения: камера получает изображения, скорректировав их на свою гамму, а дисплей в дальнейшем компенсирует эту коррекцию. И под *гамма-коррекцией* понимается не только коррекция кривой, а коррекция самого значения гаммы для правильного отображения.

Убедиться в том, что интенсивность, выдаваемая монитором, зависит от значения пикселя нелинейно, можно, рассмотрев на экране изображение с рис. 6.21.



Рис. 6.21. Проверка гаммы монитора.

Здесь квадраты с 50% серого цвета (значение пикселя 128) чередуются с квадратами, где в шахматном порядке приведены чёрные и белые пиксели (со значениями 0 и 255). Если отойти подальше от монитора, то квадраты с чёрными и белыми пикселями будут казаться серыми, так как глаз усреднит их яркость. Если монитор выдаёт интенсивность линейно, то визуально яркость всех квадратов должна примерно совпадать. В действительности на большинстве мониторов серые квадраты кажутся существенно темнее. Поменяв цвет серых квадратов так, чтобы он наилучшим образом визуально соответствовал цвету квадратов с чёрными и белыми пикселями, можно по новому значению цвета оценить гамму монитора. Следует также заметить, что на многих LCD-мониторах результат будет существенно зависеть от угла обзора.

## 6.5.2. Простейшие приемы управлениями изображением

**Линейные и нелинейные устройства.**

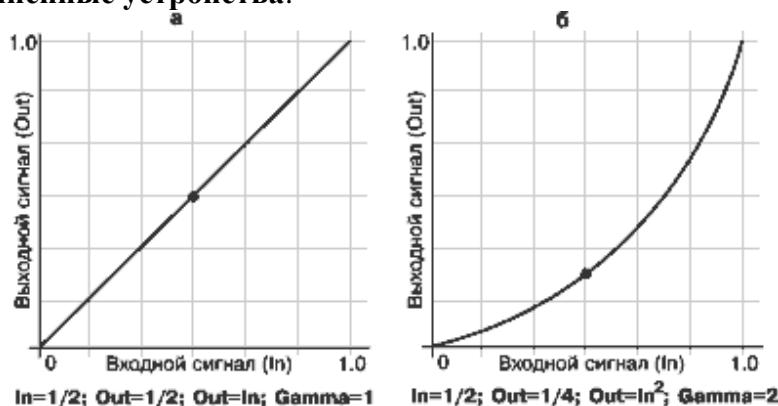


Рис. 6.22. Передача изображения в зависимости от значения гаммы

Деб: Рис менять

На рис. 6.22 приводятся 2 графика передачи входного сигнала в выходной. Слева показан график с гаммой=1, такое устройство называется *линейным* и обеспечивает простую обработку изображений. Как мы выяснили выше, нам придется иметь дело в основном с *нелинейными* устройствами, у которых гамма не равна 1

Поэтому нам часто приходится рассчитывать изображения с учетом последующего применения гаммы монитора – рис. 6.23.



Рис. 6.23. Пусть имеем изображение с гаммой=2.2 (график слева). Подвергаем изображение гамма-коррекции с гаммой=1/2.2 (график посередине). И обеспечиваем линейность передачи изображения (график справа)

Деб: Рис менять

### Яркость.

Под яркостью понимают величину общей эмиссии ЭЛТ, возникающей за счет увеличения общей величины управляющего воздействия (например, напряжения на ускоряющем электроде). На рис. 6.24 показано, каким функциям преобразования изображения соответствуют завышенная и заниженная яркости.

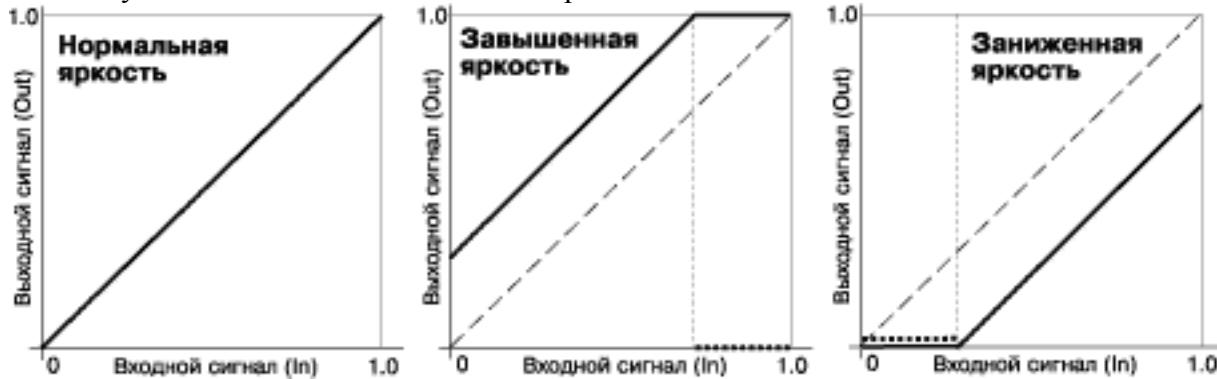


Рис. 6.24

Деб: Рис менять

### Контрастность.

Контрастность — это условная величина, обозначающая разницу между самым светлым и самым темным участком, т.е. обозначающая как бы число воспроизводимых полутонов. Для более точного понимания, в чем разница между яркостью и контрастностью, можно сказать, что яркость — это характеристика конкретного пикселя экрана, а контраст — это разница между самым ярким и самым темным пикселями. На рис. 6.25 показано, каким функциям преобразования изображения соответствуют различные значения контраста.

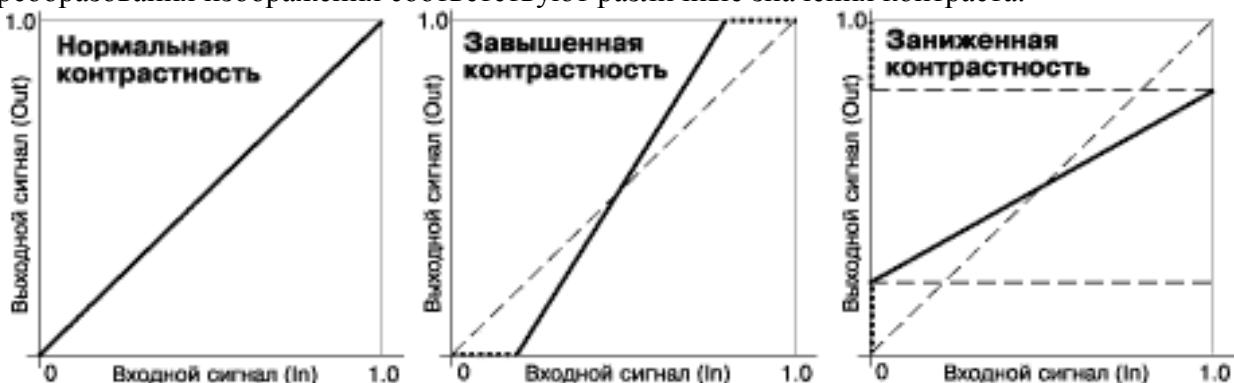


Рис. 6.25. Управление контрастом

Деб: Рис менять

Теперь ясно, как надо использовать гамма-коррекцию, яркость и контраст, для достижения требуемого эффекта при отображении изображения.

### Замечание по точности.

У нас все изображения получаются в палитре 0-255 полутоонов. Если мы применяем гамма-коррекцию, то можем потерять некоторые значения, и картина в дальнейшем будет хотя и в палитре 0-255, но с рядом неиспользуемых полутоонов. Таким образом, используя гамму, мы можем получить даже оконтуривание, т.е. на изображении будут явно проступать контура ряда областей.

*Вопрос:* почему это происходит?

Одним из решений этой проблемы является хранение изображений не в 8 битовой, а в 16 битовой палитре.

## 6.6. Понятие алиасинга

Многие артефакты, присутствующие на синтезированных изображениях, называют *алиасингом*. *Антиалиасингом* называются методы борьбы с этими артефактами. Рассмотрим пример на рис. 6.26, где приведен растеризованный отрезок.

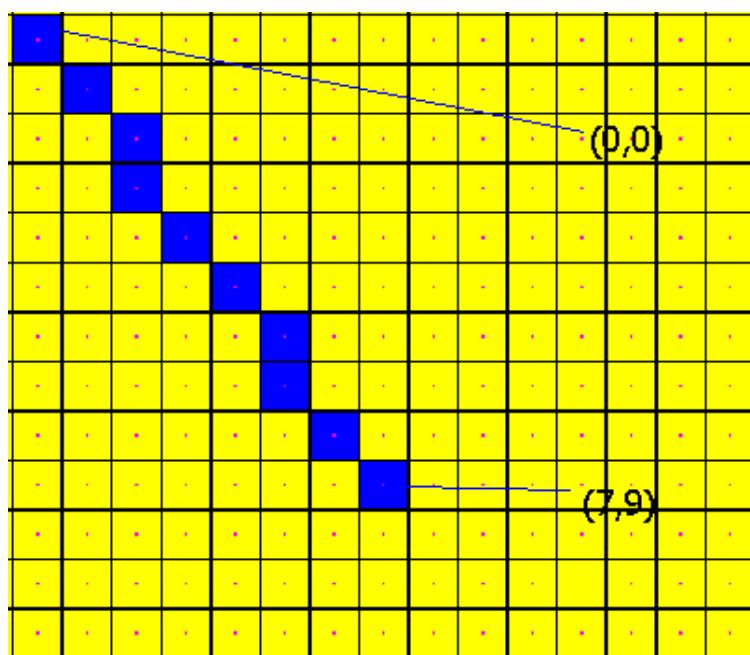


Рис. 6.26. Пиксельное представление отрезка

Деб: Рис можно не менять

Замечания к рисунку:

- Отрезок – это единый объект, гладкий. И наш глаз сразу увидит *лестничный эффект* на изображении.
- Отрезки с разными наклонами должны быть одной линейной яркости, хотя, судя по рис. 6.27: случай а) – линейная яркость 1, а в случае б) – линейная яркость в  $\sqrt{2}$  раз меньше. Зритель вполне может расценить их как отрезки, принадлежащие разным объектам, хотя они могут быть элементами границы одной машиностроительной детали.

Нам требуется провести антиалиасинг, чтобы отрезок выглядел так, как показано на рис. 6.27 случай д). На заре компьютерной графики использовался прием выравнивания линейной

яркости – в зависимости от наклона вместо одиночного пикселя ставились 2 или 4, как показано на рис. 6.27, случаи e)-f).

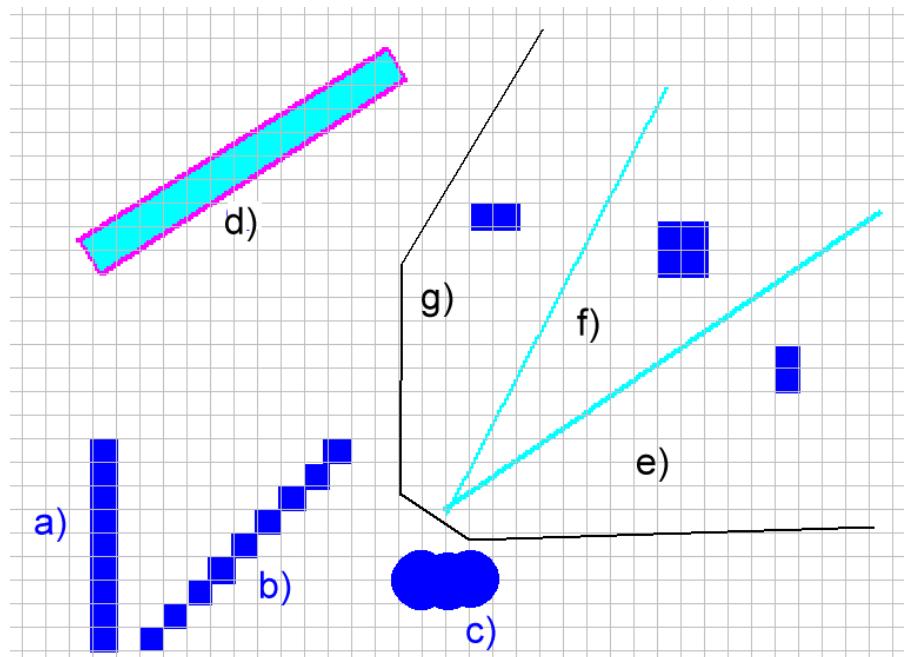


Рис. 6.27. Варианты изображения отрезков

Но лестничность все равно оставалась, поэтому применяется так называемый *субпиксельный рендеринг*. Во-первых, отрезки (и другие линии) считаются телесными, т.е. это уже не отрезок, а прямоугольник. Из рис. 6.28 видно, что:

- Если пиксель целиком принадлежит отрезку, то этому пикслю назначается 100% требуемого цвета.
- Если отрезок занимает K% от площади пикселя, то пикслю назначается K% требуемого цвета.

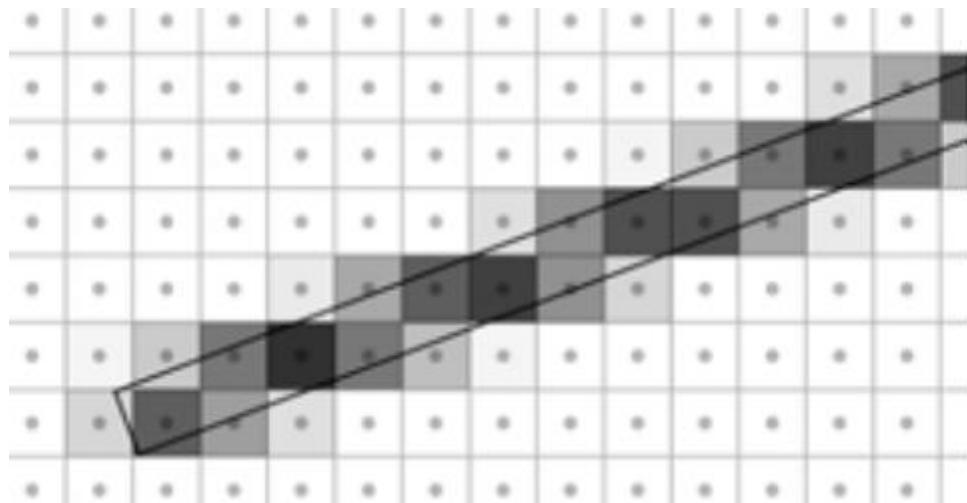


Рис. 6.28. Субпиксельный рендеринг

Реально точные проценты не подсчитываются, просто отрезок растеризуется на более мелком растре, и по количеству зачерненных пикселей определяется искомый процент. На рис. 6.29 показан пример лестничного эффекта (слева), а справа та же фигура рассчитана с помощью раstra уменьшенного всего в 2 раза.

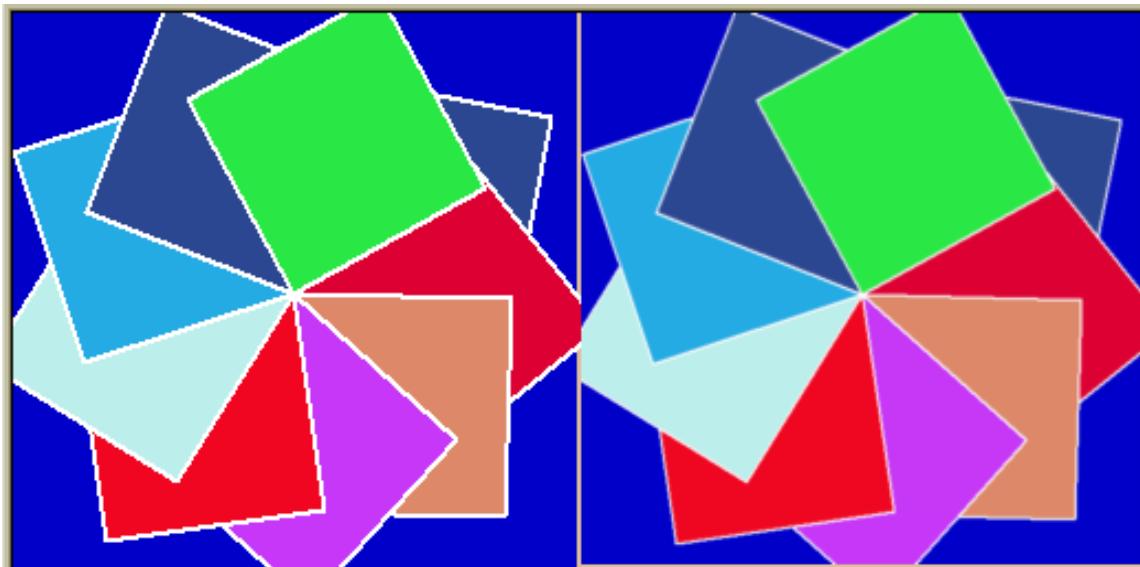


Рис. 6.29. Лестничный эффект (слева) и сглаженная фигура (справа)

Деб: Рис менять ИЛИ что с вертикальным выравниванием?

На рис. 6.30 приведен еще один пример алиасинга, который показывает, чем отличается грубый шрифт от сглаженного.

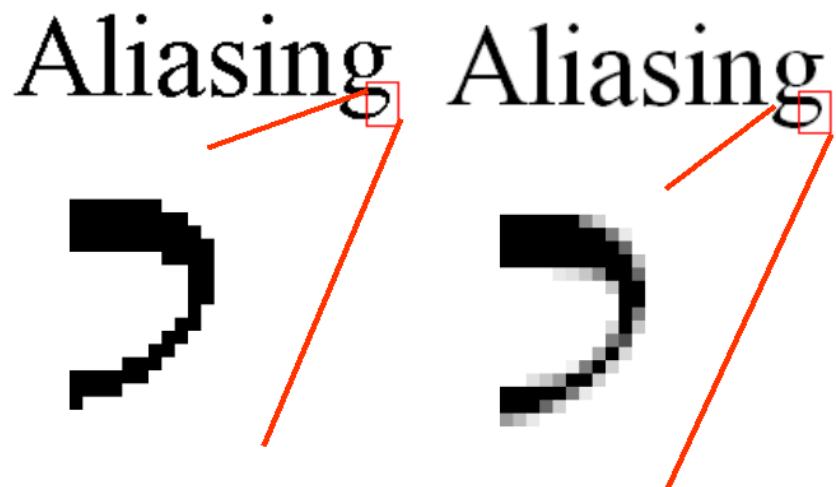


Рис. 6.30. Антиалиасинг букв

## 6.7. Алгоритм У Сяолиня

В случае растеризации отрезков, толщина которых совпадает с шириной пикселя (считаем пиксель квадратным), для субпиксельного рендеринга можно использовать алгоритм У Сяолиня (Wu Xiaolin) [10], который является модификацией алгоритма Брезенхэма. Предположим, мы хотим изобразить чёрный отрезок на белом фоне, и библиотека вывода на экран предоставляет функцию вывода пикселя `plot(x, y, color)`, где `color` задаёт яркость серого от 0 до `MAX_COLOR` (`color = 0` — чёрный цвет, `color = MAX_COLOR` — белый). Преобразуем координаты концов отрезка так же, как и в алгоритме Брезенхэма, чтобы выполнялось начало отрезка располагалось в точке  $(0, 0)$ , а конец — в точке  $(dx, dy)$ , причём  $dx \geq dy \geq 0$ . Тогда алгоритм будет выглядеть следующим образом:

```

X := 0
y := 0
err := 0
while (x < dx)
    x := x+1
    err := err+dy
    if (err > dx) then
        err := err-dx
        y := y+1
    end
    color = MAX_COLOR*err/dx
    plot(x, y, color)
    plot(x, y-1, MAX_COLOR-color)
end

```

Здесь для вычисления цвета используется целочисленное деление.

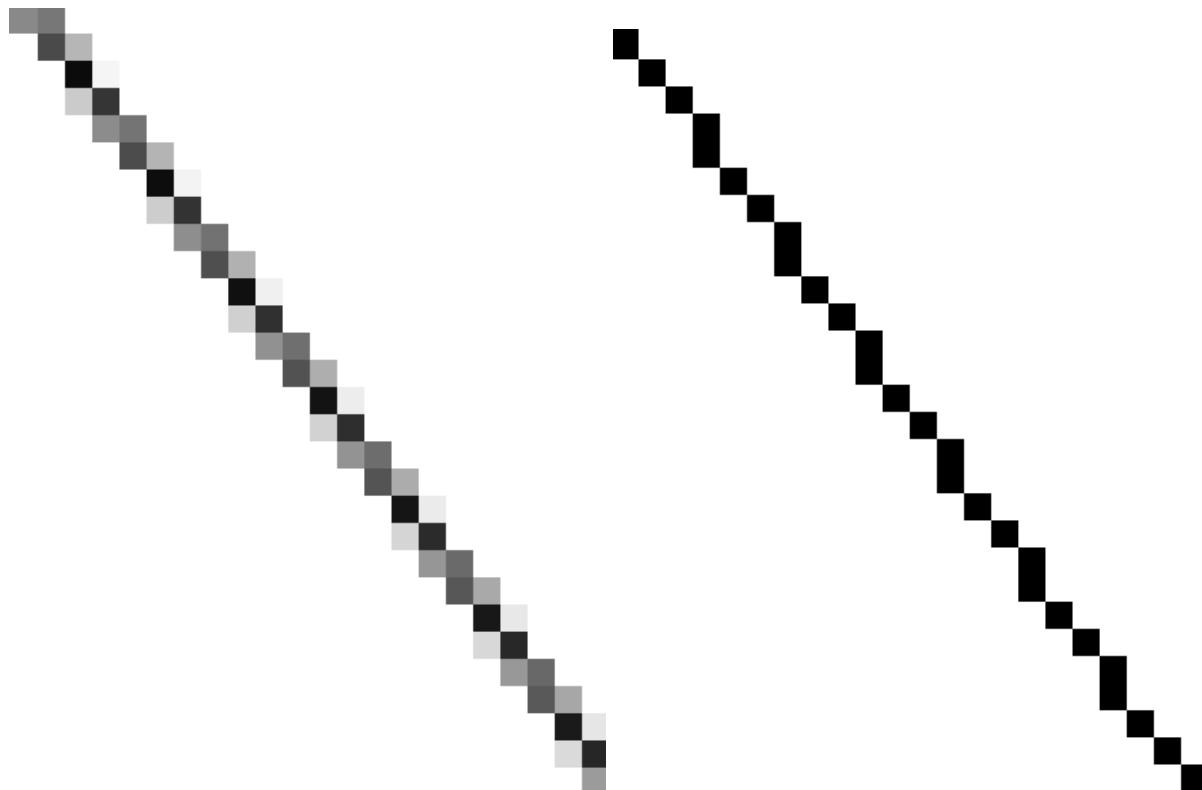


Рис. 6.31. Растеризация отрезка алгоритмом У Сюолиня (слева) и алгоритмом Брезенхэма (справа).

## 6.8. Алиасинг в анимациях

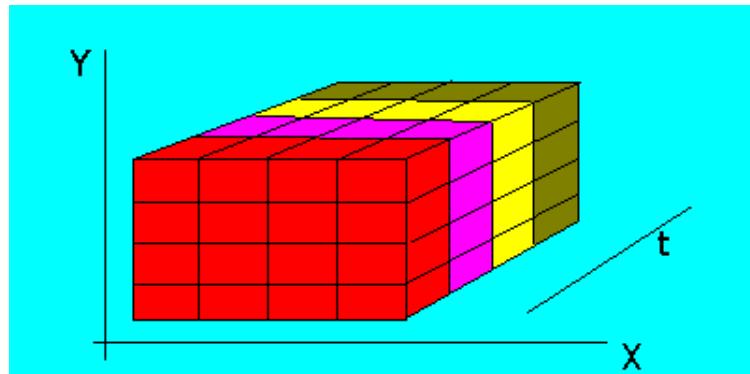


Рис. 6.32. Представление кадров анимации

На рис. 6.32 схематично представлена анимация, т.е. серия последовательных кадров фильма. Каждый отдельный кадр представлен слоем, слои идут вдоль оси времени  $t$ . Под термином *tempоральный алиасинг* понимается эффект, когда некоторая область кадра или даже отдельный пиксель на соседних кадрах имеют сильно отличающиеся значения яркости. Этакая "лестничность" по времени. Или пропадающие / появляющиеся от кадра к кадру отдельные яркие пиксели.

Если анимация получена с помощью видеосъёмки, то кадр представляет собой не плоский прямоугольник в определённый момент времени, а прямоугольный параллелепипед, толщина которого зависит от времени выдержки видеокамеры (скорости срабатывания затвора). Если время выдержки совпадает с временем между кадрами, по сути дела мы получаем темпоральный антиалиасинг: интегрирование яркости определённой области по промежутку времени. В этом случае края быстродвижущегося объекта будут размыты. Если же время выдержки очень мало, движущийся объект будет резко менять своё положение, возможно появление стробоскопического эффекта.

Когда анимация генерируется с помощью компьютера, проще всего эмулировать бесконечно малую выдержку, отображая состояние сцены в определённый момент времени. Иногда для большей реалистичности анимации эмулируют длительную выдержку, размазывая движущиеся объекты (так называемый motion blur).

## 6.9. Пиксель – это квадрат? Нет – это выборка

Деб.: Мне может тоже не нравится, но это факт. Хотя, можете предложить свои названия  
Давайте встретимся – не всё же в офф-лайне

По инерции мы думаем об изображении как о непрерывной функции яркости. Реальные изображения таковыми и являются. А пиксельные представления этих изображений – это сеточные функции, где для каждого пикселя выбирается некоторое значение, таким образом, изображение представляется так, как это показано на рис. 6.33.

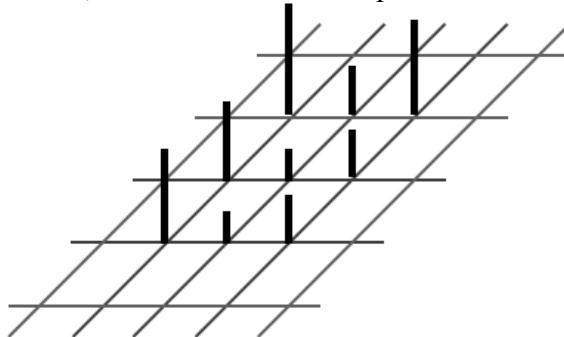


Рис. 6.33. Сеточное представление изображения

Получается, что об изображении – непрерывной функции двух переменных  $g(x, y)$  – мы имеем информацию только в сеточных точках:  $g(i, j)$ ,  $i = 0..n$ ,  $j = 0..m$ . Чтобы различать эти изображения – реальное и дискретное – последнее будем называть *дисплеем*.

Рассмотрим точное определение термина "алиасинг". Alias – псевдоним. Как известно, для представления сигнала с помощью выборки необходимо, чтобы частота выборки как минимум вдвое превышала частоту сигнала (частота Найквиста). В результате недостаточной выборки (undersampling) высокочастотный сигнал может выдавать себя за сигнал более низкой частоты. Возможное решение проблемы алиасинга – принудительное понижение частоты исходного сигнала. Что же это значит? Рассмотрим на примере одномерного сигнала, см. рис. 6.34

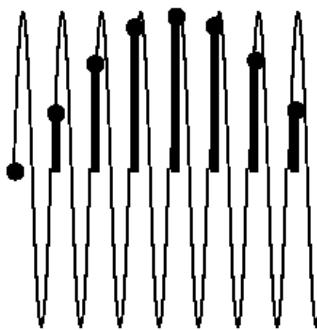


Рис. 6.34. Выборки из сигнала

Сигнал – это чистая синусоида некоторой частоты  $\omega_i$ . Если мы случайно выбрали частоту оценивания сигнала  $\omega_s < \omega_i$ , то оцифрованный сигнал кажется нам тоже синусоидой, но частоты намного меньшей, чем  $\omega_i$ . Представим, что мы также неудачно взяли частоту выборки из изображения. В результате мы на дисплее видим лестничный эффект, оконтурирования и т.п.

### 6.9.1. Регуляризация функций, сглаживание изображений

Посмотрим, как можно сгладить дисплей. Для этого обратимся к понятию *регуляризации функций*. Итак, пусть у нас имеется разрывная функция одной переменной (рис. 6.33, слева), сделаем ее непрерывной. Одно из возможных решений – пересчитать функцию по формуле

$$g_\sigma(x) = \frac{1}{\sigma} \int_{x-\frac{\sigma}{2}}^{x+\frac{\sigma}{2}} g(u) du. \text{ На рис. 6.33 красным показано, как мы сделали непрерывность.}$$

Аналогичный прием, примененный к функции с разрывной производной (рис. 6.33. справа), дает гладкую кривую.

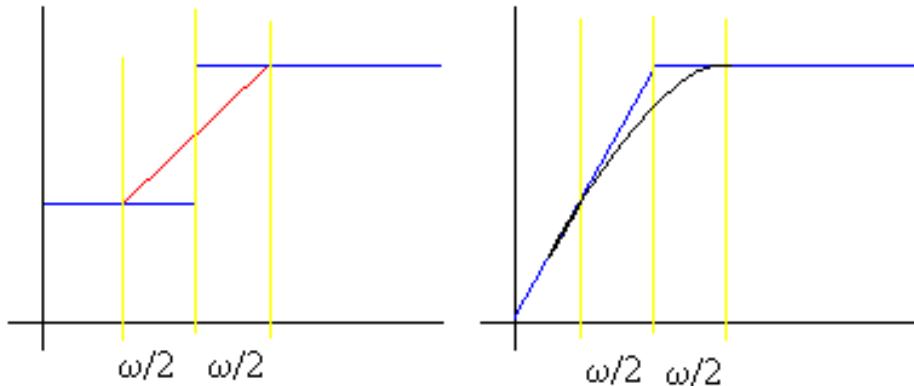


Рис. 6.33. Регуляризация функций  
Желтый поменять, т.к. это не слайд

Аналогичную регуляризацию можно применить и на плоскости – для непрерывной функции двух переменных.

$$g_\sigma(x, y) = \frac{1}{A_\sigma} \iint_{\sigma(x, y)} g(u, v) \cdot du \cdot dv,$$

здесь  $\omega(x, y)$  – некоторая окрестность точки  $(x, y)$ , а  $A_\sigma$  – площадь этой окрестности. Для сглаживания изображений применим данный подход к дисплею – дискретному изображению. Поскольку лучше иметь немного размытое изображение  $g_\sigma$ , чем пугающее "лестничное" с контурами  $g(i, j)$ ,  $i = 0..n, j = 0..m$ .

$$g_\sigma(i, j) = \frac{1}{A_\sigma} \sum_{(k, l) \in \sigma(i, j)} g(k, l).$$

здесь  $\omega(i, j)$  – некоторая окрестность пикселя  $(i, j)$ , а  $A_\sigma$  – число пикселей в этой окрестности. Наиболее часто в качестве окрестности пикселя берется бокс (прямоугольник) с размерами  $(2b+1)(2h+1)$  с центром в рассматриваемом пикселе, тогда вычисление  $g_\sigma$  производится по формуле:

$$g_\sigma(i, j) = \frac{1}{(2b+1)(2h+1)} \sum_{-b \leq k \leq b} \sum_{-h \leq l \leq h} g(i+k, j+l).$$

Этот объект называется *бокс-фильтром* и записывается в виде матрицы, например ,

$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Данный бокс-фильтр считает вклад каждого пикселя в результат равнозенным. Существуют и другие сглаживающие бокс-фильтры:

- $\frac{1}{10} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$  – здесь значению самого рассматриваемого пикселя дается важность в 2 раза больше, чем остальным.
- $blur = \frac{1}{6} \begin{bmatrix} 1 & 1 \\ 1 & 2 & 1 \\ 1 & & \end{bmatrix}$  – этот фильтр называется фильтром размытия. Пустые места в матрице – это нули.

- $gauss = \frac{1}{74} \begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 4 & 5 & 4 & 2 \\ 3 & 5 & 6 & 5 & 3 \\ 2 & 4 & 5 & 4 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix}$  – гауссов фильтр.

**Замечание.** Как правило, материал, посвященный фильтрации, излагается с опорой на Фурье-анализ. Нам удалось обойтись без этого согласно поставленным целям.

### 6.9.2. Прореживание или фильтрация

Часто возникает задача уменьшить разрешение дискретного изображения, т.е. пересчитать его на более грубую сетку. Один из наиболее оперативных способов применяемых в редакторах является *прореживание*, когда определенное число столбцов и строк выбрасывается из изображения, рис. 6.34, слева. Как видно из рис. 6.34, справа, удовлетворительный результат получается, если в выходное прореженное изображение помещать значение, полученное от применения сглаживающего фильтра.

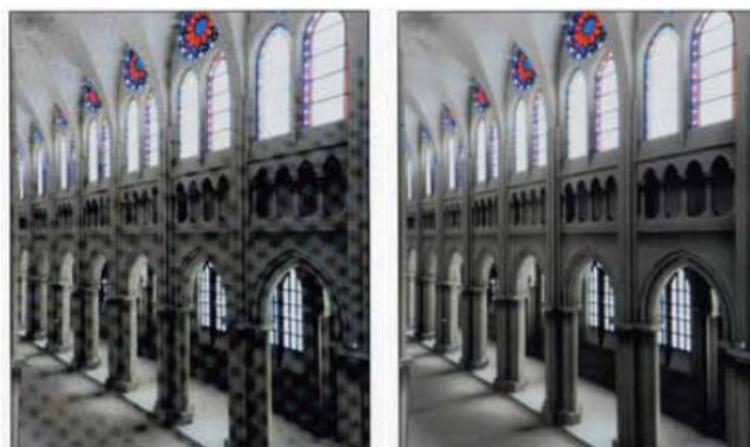


Рис. 6.34. Слева: прореживание, справа: применение гауссова фильтра **ВЗЯТ У КОГО\_ТО -  
найти**

### 6.10. Задачи обработки изображений

Наиболее популярные задачи:

1. Устранение дефектов изображения (напр., устранение снега на фотопленке).
2. Улучшение изображения (напр., сделать темнее недодержанную фотографию).
3. Упрощение изображения (напр., цветное, черно-белое, каркасное).

Во многих из них требуется получить очерки изображенных объектов для того, чтобы по ним построить модель объекта и распознать объект. Когда хотят найти на изображении танк, его цвет не важен. На рис. 6.35 показано как можно было бы из изображения извлечь структурную информацию:

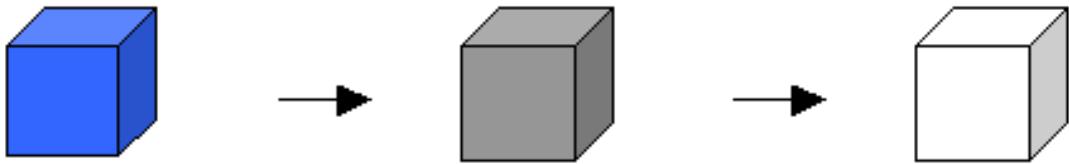


Рис. 6.35. Этапы обработки изображения

- Сначала цветное изображение переводится в черно-белое.
- Затем из черно-белого изображения выделяются характерные линии.

### 6.10.1. Оператор Робертса

Приступим к рассмотрению оператора, который может обозначаться множеством названий: градиентное изображение, пространственное дифференцирование, выделение контуров, выделение границ, повышение резкости, взятие градиента. Рассмотрим наше дискретное изображение  $g(i, j)$ ,  $i = 0..n, j = 0..m$ . Оценка градиента в пикселе  $(i, j)$  делается на основе значений трех соседей (рис.6.36):

$$\|\nabla g(x, y)\| \approx R(i, j) = \sqrt{[g_{i,j} - g_{i+1,j+1}]^2 + [g_{i,j+1} - g_{i+1,j}]^2}, \quad g_{ij} = g(i, j).$$

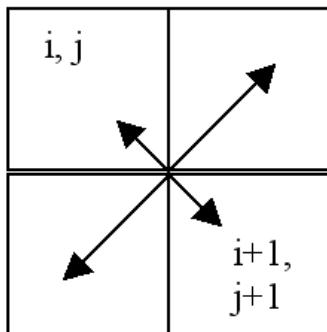


Рис. 6.35. Оценка градиента

Очевидно, что в областях с однородной интенсивностью  $R(i, j) = 0$ .

Здесь мы видим много сложных вычислений, что неприемлемо, поскольку нам нужен быстрый алгоритм, способный на лету обрабатывать снимки, поступающие со спутников. Опираясь на неравенство:

$$R(i, j) \leq F(i, j) = |g_{i,j} - g_{i+1,j+1}| + |g_{i,j+1} - g_{i+1,j}| \leq \sqrt{2}R(i, j)$$

получаем более быстрый инструмент  $F(i, j)$  (оператор Робертса), обеспечивающий аналогичный результат. Он применяется следующим образом:

```
if (F(i, j) > C)
    (i, j) := 1;
else
    (i, j) := 0;
```

$C$  – это некоторый подбираемый порог, который позволяет игнорировать пиксели с незначительными отклонениями значений, но не пропустить границу области. На рис. 6.36 приводятся два градиентных портрета, полученные из одного изображения при разных значениях порога.

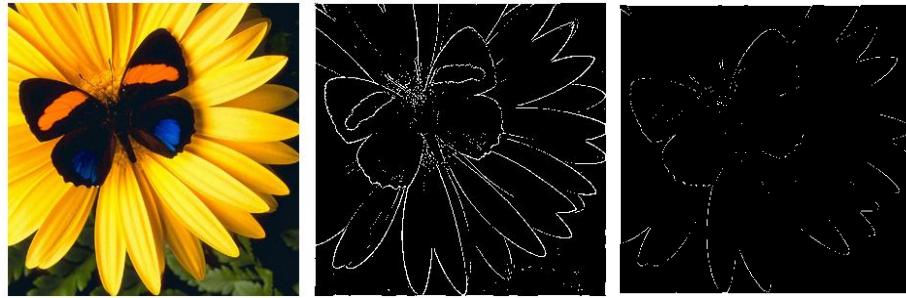


Рис. 6.36. Исходное изображение и его градиентные портреты ( $C=40$  и  $C=200$ )



Рис. 6.37. Исходное изображение и его градиентный портрет (оператор Собеля)

### 6.10.2. Оператор Собеля

Оператор Собеля использует другую формулу для оценки градиента. Если обозначить

пиксель и его 8 окружающих соседей в виде матрицы  $\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$ , то формулы выглядят

следующим образом:

$$S_x = (c + 2f + i) - (a + 2d + g),$$

$$S_y = (g + 2h + i) - (a + 2b + c),$$

$$S = \sqrt{S_x^2 + S_y^2}.$$

Делаем аналогичное упрощение и получаем счетную формулу:

$$S \approx |S_x| + |S_y|.$$

Пример применения оператора Собеля см. на рис. 6.37.

### 6.10.3. Выделение контуров

Кроме двух рассмотренных популярных операторов можно применять и бокс-фильтры, которые имеют отрицательные элементы. Рассмотрим на примере. Пусть у нас бокс-фильтр

$3 \times 3$ :  $h(u, v) = \frac{1}{4} \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ . В примере числа подобраны специальном образом, поэтому

можно использовать коэффициент  $1/4$ . Как правило, дифференцирующие бокс-фильтры используются вместе с порогом, как и оператор Робертса. На рис. 6.38 слева дано исходное изображение в виде значений пикселей.

|   |   |   |   |   |    |    |    |    |
|---|---|---|---|---|----|----|----|----|
| 0 | 0 | 0 | 0 | 0 | 25 | 25 | 25 | 25 |
| 0 | 0 | 0 | 0 | 0 | 25 | 25 | 25 | 25 |
| 0 | 0 | 0 | 0 | 0 | 25 | 25 | 25 | 25 |
| 0 | 0 | 0 | 0 | 0 | 25 | 25 | 25 | 25 |
| 0 | 0 | 0 | 0 | 0 | 25 | 25 | 25 | 25 |
| 0 | 0 | 0 | 0 | 0 | 25 | 25 | 25 | 25 |
| 0 | 0 | 0 | 0 | 0 | 25 | 25 | 25 | 25 |
| 0 | 0 | 0 | 0 | 0 | 25 | 25 | 25 | 25 |
| 0 | 0 | 0 | 0 | 0 | 25 | 25 | 25 | 25 |

**a**

|   |   |   |   |    |    |   |   |   |
|---|---|---|---|----|----|---|---|---|
| 0 | 0 | 0 | 0 | 25 | 25 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 25 | 25 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 25 | 25 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 25 | 25 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 25 | 25 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 25 | 25 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 25 | 25 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 25 | 25 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 25 | 25 | 0 | 0 | 0 |

**b**

Рис. 6.38. Исходное изображение и результат применения фильтра **ПЕРЕРИСОВАТЬ**

Применяем фильтр  $h$  по формуле:

$$b[x, y] = \sum_{u=-1}^{+1} \sum_{v=-1}^{+1} h[u, v] a[x-u, y-v]$$

В результате мы получили ненулевые только пиксели границы. Отметим, что данный фильтр сконструирован для выделения только вертикальных контуров. Другие фильтры выделения контуров и границ:

- $h = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$  – может оставлять изолированные точки.
- $sharp = \begin{bmatrix} -1 & & \\ -1 & 5 & -1 \\ & -1 & \end{bmatrix}$  – фильтр увеличения резкости.
- $h = \begin{bmatrix} +1 & -2 & +1 \\ -2 & 5 & -2 \\ +1 & -2 & +1 \end{bmatrix}$ .
- $cont = \begin{bmatrix} -1 & & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix}$  – выделение контура.

Каждый из этих фильтров работает хорошо в тех или иных условиях, т.е. применяется для изображений определенного вида.

Вернемся к рис. 6.27, где демонстрировался лестничный эффект, и прямо к этому рисунку применим выделение контуров – см. рис. 6.39.

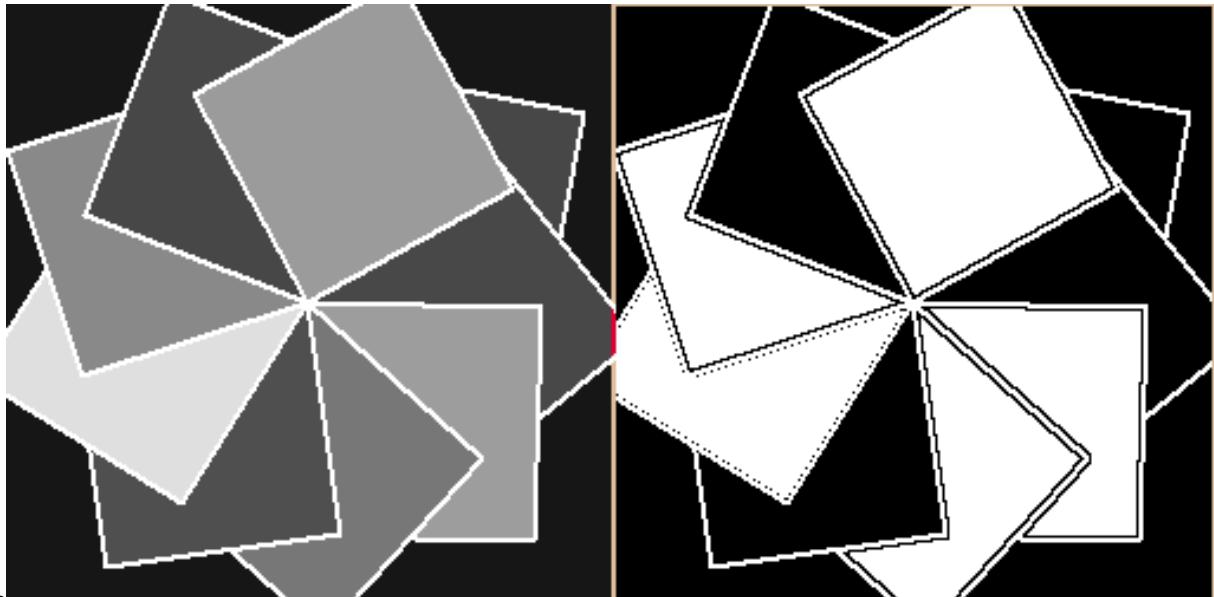


Рис. 6.39. Выделение контуров. Слева: исходное изображение, переведенное в черно-белое  
**НЕГАТИВ**

На рис. 6.39, справа видно, что оператор Робертса по-разному работает на разных углах наклона границ.

## 6.11. Замечания о цветных изображениях

### 6.11.1. Перевод в черно-белое

Такой перевод, как правило, выполняется перед операцией выделения границ. Если мы имеем полноцветное 24 битовое изображение с компонентами (каналами) RGB, то для получения черно-белого изображения необходимо применить формулу:

$$Y = 0.299R + 0.587G + 0.114B.$$

Мы получаем результат аналогичный тому, когда по черно-белому ТВ смотрим цветной фильм. Из формулы видно, что самым информативным является зеленый канал, а самым неинформативным – синий.

### 6.11.2. Дизеринг и сглаживание цветных

В данной главе мы рассматривали алгоритмы применительно к ахроматическим полутоновым изображениям. В случае операций дизеринга и сглаживания достаточно 3 раза повторить алгоритм для красного, синего и зеленого каналов независимо.

Алгоритмы дизеринга с диффузией ошибки часто применяются для перевода полноцветных 24 битовых изображений в 8 битовые. Как разделить эти 8 битов между тремя цветами? Ответ прост:

- Красный – 3 бита,
- Зеленый – 3 бита,
- Синий – 2 бита, т.к. он наименее информативен.

### 6.11.1. Несколько примеров, акварелизация, тиснение

На рис. 6.40, 6.41 приведены результаты обработки одного изображения разными бокс-фильтрами. **РИСУНКИ НИЖЕ НЕ НАШИ**



Рис. 6.40. Размытие бокс-фильтрами blur (слева), и gauss (справа)



Рис. 6.41. Увеличение резкости фильтром sharp и выделение контура фильтром cont

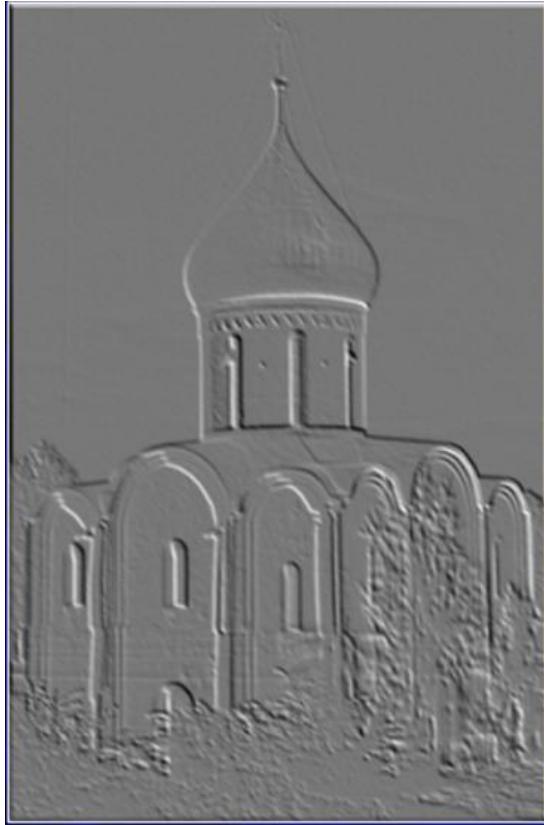


Рис. 6.42. Тиснение

**Тиснение.** Это сложная операция, состоящая из нескольких шагов. Тиснение делается почти также как размывание и увеличение резкости. Процесс начинается с обычным цветным изображением.

- Каждый пиксель в изображении обрабатывается ядром тиснения размером 3x3 –  
$$emboss = \begin{bmatrix} -1 & & \\ 1 & 0 & -1 \\ & 1 & \end{bmatrix}$$
. В отличие от ядер размывания и резкости, в которых сумма коэффициентов равна 1, сумма весов в ядре тиснения равна 0. Это означает, что "фоновым" пикселям (пикселям, которые не находятся на границах перехода от одного цвета к другому) присваиваются нулевые значения, а не фоновым пикселям – значения, отличные от нуля.
- После того, как значение пикселя обработано ядром тиснения, к нему прибавляется 128. Таким образом, значением фоновых пикселей станет средний серый цвет (красный = 128, зеленый = 128, синий = 128). Суммы, превышающие 255, можно округлить до 255 или взять остаток по модулю 255, чтобы значение оказалось между 0 и 255.
- В тисненом варианте изображения, контуры кажутся выдавленными над поверхностью. Направление подсветки изображения можно изменять, меняя позиции 1 и -1 в ядре. Если, например, поменять местами значения 1 и -1, то реверсируется направление подсветки.

**Акварелизация.** Это также сложная операция, состоящая из нескольких шагов.

Акварельный фильтр преобразует изображение, и после обработки оно выглядит так, как будто написано акварелью, например, цифровое изображение, просканированное с фотографии.

- Первый шаг в применении акварельного фильтра – сглаживание цветов в изображении. Одним из способов сглаживания является процесс *медианного осреднения* цвета в каждой точке (медианный фильтр – 3x3 или 5x5). Значение цвета каждого пикселя и его 24 (8) соседей помещаются в список и сортируются от

меньшего к большему. Медианное тринадцатое (пятое) значение цвета в списке присваивается центральному пикселью.

- После сглаживания цветов, компьютер обрабатывает каждый пиксель в изображении ядром резкости, чтобы выделить границы переходов цветов.
- Результатирующее изображение напоминает акварельную живопись. Это лишь один пример, который показывает, как можно объединять различные методы обработки изображений и добиваться необычных визуальных эффектов.

Упражнение. Построить оператор акварелизации.

## 6.12. Прозрачность

### 6.12.1. Альфа канал

В самом начале курса мы говорили о том, что у станции Silicon Graphics в буфере кадра цвета описываются 32-мя разрядами. Из них 24 разряда – это полный цвет RGB, а оставшийся байт – это 4-й, так называемый альфа-канал. Этот канал содержит значение непрозрачности цвета  $opacity$  от 0 до 255. В расчетах считается, что непрозрачность равна  $\frac{opacity}{255}$  от 0 до 1.

Итак, пусть мы отображаем в пикселе первый объект A, который характеризуется цветом A и непрозрачностью  $\alpha$ . Считая, что пиксель квадрат, выражение RGBA означает, что доля  $\alpha$  всей площади пикселя покрыта полностью непрозрачным объектом цвета A. Поскольку мы усредняем все, попавшее в телесный угол, соответствующий пикслю, то в пикселе будет средний цвет  $\alpha \cdot A$ .

Аналогично, если в пиксель поместим один объект B с цветом B и непрозрачностью  $\beta$ , то считаем, что доля  $\beta$  пикселя занята непрозрачным цветом B. А пиксель будет показывать усредненный цвет  $\beta \cdot B$ .

Теперь посмотрим, как второй объект добавляется к первому. Из геометрических рассмотрений из-за верхнего объекта B, только доля  $(1 - \beta)$  пикселя может быть видима, т.е. через долю  $(1 - \beta)$  пикселя просматривается цвет, созданный первым объектом A –  $\alpha \cdot A$ . Исходя из таких рассуждений, получаем окончательный цвет пикселя –  $\beta B + (1 - \beta)\alpha A$ .

### 6.12.2. Композиция изображений

Задача: имеется 2 изображения  $In1$  и  $In2$ . Требуется выполнить их композицию на основе значений прозрачности – получить изображение  $Out$ .

Положим

$$Out(x, y) = (1 - \alpha(x, y)) \cdot In1(x, y) + \alpha(x, y) \cdot In2(x, y).$$

Используя эту формулу легко написать псевдокод анимации из  $n$  кадров со следующим сценарием:

- Сначала в кадре  $In1$
- Затем через него начинает пропасть  $In2$
- В конце в кадре  $In2$

Псевдокод.

$$\text{for } \alpha = 1; \alpha <= 1; \alpha = \alpha + \frac{1}{n}$$

$$Out(x, y) = (1 - \alpha(x, y)) \cdot In1(x, y) + \alpha(x, y) \cdot In2(x, y)$$

По такому сценарию мы видим, как на экране ТВ одно лицо преобразуется в другое.

### 6.12.3. Синяя комната

Метод синей комнаты – это еще один метод композиции нескольких изображений. Основным понятием является *ключевой цвет* или диапазон цветов. Представим, что вам надо снимать фильм примерно такого сценария, как показан на рис. 6.43.



Рис. 6.43. Шторм

Вообще-то такую сцену создать очень дорого, особенно в павильоне. Рассмотрим, как это делалось в реальности.

- 1-й компонент. Рис. 6.44 – кто-то, когда-то заснял шторм, самое око тайфуна. Режиссер взял это из фильмотеки.



Рис. 6.44. Око тайфуна

- 2-й компонент. Рис. 6.45 – на кладбище забытых кораблей оператор снял ролик, создавая движением камеры иллюзию корабля, плывущего в шторм на фоне синего неба.

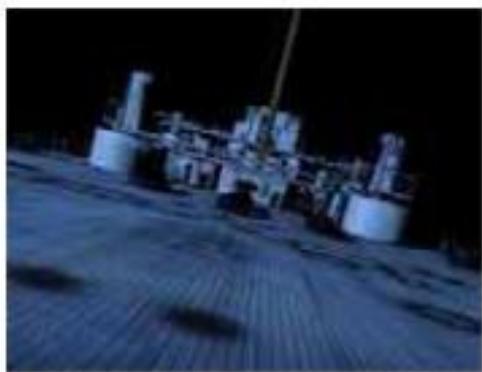


Рис. 6.45. Старый корабль

- 3-й компонент. Рис. 6.46 – актеры заднего плана изображают страдания людей, терпящих бедствие в шторм. Это самый дорогой технически ролик, т.к. в студии пришлось воспроизвести палубу корабля.



Рис. 6.46. Сцена страданий заднего плана

- 4-й компонент. Рис. 6.47 – актер первого плана играет отчаяние и усилия, возможно сидя на стуле.



Рис. 6.47. Страдания первого плана

После того как все частные ролики получены, осталось подогнать их под единый масштаб и наложить друг на друга согласно введенной нами последовательности. Все кадры прямоугольные. Для того, чтобы со следующего кадра бралась только необходимая информация, применяется операция кининга (keying), когда ключевой цвет считается прозрачным и все пиксели, содержащие такой цвет просто игнорируются. У 2-го компонента ключевые цвета – оттенки синего. У 3-го и 4-го – оттенки зеленого. Первоначально в этой технологии использовался в основном синий цвет, как наименее информативный – отсюда и ее название "синяя комната". В последнее время зеленый цвет для незначащего фона используется очень часто.

## 7. Визуализация в научных вычислениях

Картина богаче тысячи слов

Известная поговорка, см.

Wikipedia, the free encyclopedia:

*A picture is worth a thousand words* is a familiar proverb that refers to the idea that complex stories can be told with just a single still image, or that an image may be more influential than a substantial amount of text. It also aptly characterizes the goals of visualization where large amounts of data must be absorbed quickly.  
([http://en.wikipedia.org/wiki/A\\_picture\\_is\\_worth\\_a\\_thousand\\_words](http://en.wikipedia.org/wiki/A_picture_is_worth_a_thousand_words))

Представление в графическом виде исходных данных и результатов расчетов (Presentation Graphics) является одним из основных приложений компьютерной графики. В конце 1970-х в США была даже выдвинута стратегическая инициатива ViSC (Visualization in Scientific Computing). Во многих современных курсах по компьютерной графике на эту сторону компьютерной графики часто не обращают внимания. Рассмотрению вопросов научной визуализации можно посвятить целый курс, и мы уделим определенное внимание этим вопросам, чтобы познакомиться с основными моментами этой дисциплины. Для начала обратимся к математическому энциклопедическому словарю [9], чтобы заодно вспомнить и школьный курс, что понималось под термином "график функции" несколько десятилетий назад, в пору ручного их построения.

*График функции* – это множество точек плоскости с прямоугольными координатами  $(x, y)$ , где  $y = f(x)$ ,  $x \in E$  (область определения данной функции). Для построения графика функции обычно изучают ее следующие свойства:

- Область определения.
- Интервалы непрерывности.
- Интервалы существования и непрерывности первых производных, вторых производных.
- Интервалы монотонности (по знакам производных).
- Поведение функции при стремлении аргумента к границе  $E$ .
- Точки экстремума, перегиба.

Теперь для построения графика добавляем еще РЯД ТОЧЕК в зависимости от ТОЧНОСТИ построения графика.

Эти архаичные правила построения графика достаточно трудно применить в общем случае, ведь одна из основных задач компьютерной графики – основная задача научной визуализации – это представление в графическом виде функциональных зависимостей вида

$$z = f(x_1, x_2, \dots, x_n).$$

Если  $z$  является вектором  $(z_1, z_2, \dots, z_k)$ , то будем говорить о совместном графическом представлении  $k$  функций.  $f$  может быть сеточной функцией или задаваться некой вычислительной процедурой. Вполне понятно, что выяснение характеристик графика, приведенных выше, может оказаться невозможным или очень трудоемким. Компьютерная графика чаще предлагает пользователю возможность построить график и уже по нему

анализировать поведение функции. Правда, область  $E$  надо знать до построения, иначе могут быть неприятности при вычислениях.

В конечном итоге – это задача компьютерной графики показать функцию, и она нужна не только для того, чтобы исследователь нарисовал ее и показал другим. Он, может быть, сам увидит ее впервые.

## **7.1. Графики для презентаций**

В нашей повседневной жизни мы часто визуализируем различные числовые данные. В компьютере мы имеем значения следующих типов: вещественные, целые, перечислимые. Из них образуются различные числовые структуры, массивы. Для графического представления отдельных числовых значений выбираются различные линейные, плоские или трехмерные фигуры, а их сравнение изображается при помощи длины, площади или объема этих фигур. Изображаются также и зависимости, например, график осадков по месяцам и т.п. В целом арсенал представления таких зависимостей достаточно богат, примеры можно увидеть даже в MS Office (Word, PowerPoint и Excel). В связи с преимущественной областью использования таких графических представлений этот арсенал графических средств часто называют *бизнес-графикой*.

## **7.2. Визуализация расчетных данных**

Понятие научных исследований очень широкое. В каждой области имеются свои традиционные графические представления, например, в генетических исследованиях применяются трехмерные изображения моделей молекул – открытая программа Rasmol (<http://www.openrasmol.org/>) и другие. Здесь мы рассмотрим только базовые наиболее признанные и часто употребляющиеся графические представления, которые применяются при решении расчетных задач. Для примера можно обратиться к таким известным программным продуктам, как Matlab (<http://www.mathworks.com>), Mathematica (<http://www.wolfram.com>), Maple (<http://www.maplesoft.com>), MathCAD (<http://www.mathsoft.com>) и другие, см. также [13]. Особо отметим графические программные продукты PW-Wave фирмы Visual Numerics (<http://www.vni.com/>), DX и OpenDX фирмы IBM (<http://www.research.ibm.com/dx/>).

В начале данного раздела мы рассмотрели типовую процедуру построения графика. Она, в конечном итоге, приводит к некоторому набору точек плоскости, которые в дальнейшем необходимо соединить линиями согласно тем или иным правилам. Чаще всего соединяем отрезками прямых. Можно применить какую-нибудь интерполяцию. Подробная иллюстративная часть в данный конспект не вошла, для ознакомления можно посетить упомянутые сайты Matlab и др.

### **7.2.1. Одномерные графики**

Одномерные графики применяются для графического представления функций следующего вида:

- Функция одного переменного  $y = f(x)$ .
- Неявная функция  $F(x, y) = 0$ .
- Параметрическое задание  $x = x(t)$ ,  $y = y(t)$ .

Для построения графика вычисляется множество точек плоскости  $\{(x_i, y_i)\}_{i=1}^n$ , для изображения применяются декартова или полярная система координат.

Применяется одновременное изображение нескольких функций. Для того, чтобы различать графики различных функций используются *графические атрибуты*: цвет, тип линии (сплошная, пунктирная и т.д.), толщина линии.

Для повышения информативности график снабжается *аннотирующими элементами*: изображение и оцифровка координатных осей, масштабные линейки и *легенды*. Легенда – это графическая таблица, на которой дано соответствие конкретного графика набору графических атрибутов.

Отметим существенную разницу между одномерным графиком, который строится средствами бизнес-графики (например, в MS Word), и одномерным графиком, построенным в какой-либо системе научной визуализации (см. рис. 7.1). Здесь для примера взяты следующие данные:

|              |     |       |     |
|--------------|-----|-------|-----|
| X[]: 0.23547 | 5   | 12.33 | 48  |
| Y[]: 4.2     | 2.1 | 0.5   | 3.0 |

С точки зрения исследователя-вычислителя график данной функции – это, например, ломаная, специфицированные точки которой соединены отрезками прямых, см. рис. 7.1. Построить аналогичный график в MS Word не представляется возможным.

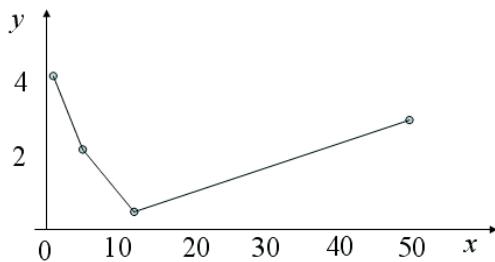


Рис. 7.1. Одномерный график: научная визуализация

Одной из задач компьютерной графики является разработка автоматического алгоритма проставления подписей или числовых меток около изображений координатных осей. Очевидно, что для приведенного выше графика подписи около оси X, состоящие из 4 чисел {0.23547, 5, 12.33, 48} приведут к плохой "читабельности" графика. Куда удобнее промаркировать как {0, 10, 20, 30, 40, 50} или {0, 5, 10, ..., 45, 50}. Как правило, если в системе научной визуализации подобный алгоритм дает неудовлетворительную последовательность числовых меток, то пользователь имеет возможность задать их принудительно. Простановка числовых меток под графиками является частной задачей более широкой постановки, называемой задачей простановки меток (*map labeling*), которая в конце прошлого века сообществом разработчиков ГИСов была определена как важная проблема (*challenge*), требующая решения.

Кратко можно сказать, что в бизнес графике подписи под осями – это не числовые метки, проставляемые в нужных местах согласно принятому масштабу, а, скорее это имена этих точек. И аналогично тому, как MS Office может автоматически сократить "Январь" до "Янв", он числовые метки "100" и "123.48" может сократить до "100" и "123".

### 7.2.2. Двумерные графики

Двумерные графики применяются для графического представления функций следующего вида:

- Функция двух переменных  $z = f(x, y)$ .
- Неявное задание  $F(x, y, z) = 0$ .
- Параметрическое задание  $x = x(u, v), y = y(u, v), z = z(u, v)$ .

Для построения графика вычисляется множество точек в пространстве  $\{(x_i, y_i, z_i)\}_{i=1}^n$ , применяются декартова, цилиндрическая или сферическая системы координат. Решается дополнительная задача проецирования графика на плоскость изображения. Отметим наиболее часто применяемые графические представления упомянутых зависимостей:

- Использование методов графического представления функций одной переменной – "понижение размерности" данных. Для этого либо фиксируется одна из переменных, либо используется сложная зависимость – связывание параметров, например,  $y = q(x)$ , тогда задача сводится к построению графика функции  $z = f(x, q(x)) = \bar{f}(x)$ . Таким образом, мы можем строить и изучать след функции на интересующей нас кривой.
- Изображение функций в виде поверхностей: проволочное, каркасное с удалением невидимых линий, на основе очерковых или силуэтных линий, полутонаовые и цветные изображения поверхностей.
- Изображение изолиниями  $f(x, y) = const$  и цветотоновыми картами. Легенда, как правило, определяет соответствие цветов значениям функции.
- Совместное изображение нескольких функций. Для совместного графического представления двух функций чаще всего применяются векторные поля в виде полей скоростей на плоскости. Для совместного представления трех функций – векторные поля в пространстве.
- Ряд библиотек и программных продуктов позволяет строить *линии тока* дополнительно к векторным полям. Здесь следует отметить, что линия тока – это решение дифференциального уравнения, определяемого выбранной математической моделью процесса течения. Векторное поле – это начальные условия для этого уравнения. И начальные условия компьютерная графика позволяет изобразить. А вот рассчитывать решение должна прикладная программа, а не графическое программное обеспечение. Графическое ПО должно красиво и разборчиво построить линии тока, точки которых рассчитаны в приложении.

Аннотирующие элементы графика – это изображение и оцифровка координатных осей, масштабные линейки и легенды. Конечно, сложность нанесения подписей существенно возрастает по сравнению с одномерными графиками. Графические атрибуты (типы и толщины линий, цвет), служат для маркировки выделенных линий, например, отдельных изолиний или сечений на поверхности.

### 7.2.3. Функции трех переменных

Функции вида  $t = f(x, y, z)$ , а также неявное и параметрическое задания, – это четырехмерные объекты и каких-либо общепринятых методов графического представления таких объектов как целого не существует, поэтому применяются различные методы понижения размерности данных или связывания параметров. Наиболее известны представления:

- Семейства изоповерхностей функции, т.е.  $f(x, y, z) = const$ .
- Следы функции на сечениях поверхностями (плоскостью, цилиндром, сферой, ...).
- Следы функции на кривых (луч, дуга окружности, ...).

Для функций, зависящих более чем от 3-х переменных, как правило, применяются различные методы понижения размерности или специфические методы повышения информативности элементов изображения.

## 7.3. Визуальное представление многопараметрических данных

Функциональные зависимости, рассмотренные выше, представляются в ЭВМ как сеточные функции в виде многомерных массивов. Иногда требуется вести визуальный контроль объектов, которые описываются набором параметров, особенно, если эти параметры зависимы. Конечно, можно как в кабине пилота аэроплана завести несколько шкал и наблюдать за всеми их показаниями. В данном разделе мы рассмотрим наиболее интересные способы графического представления многопараметрических данных, которые часто применяются в статистических исследованиях.

### 7.3.1. Лица Чернова

Для контроля параметров работы атомной электростанции Чернов [14] предложил оригинальный способ, который получил название *Лица Чернова*. Изложим кратко идею этого метода. Пусть наша система описывается набором в общем случае зависимых параметров  $P = \langle p_1, p_2, p_3, \dots, p_n \rangle$ . В случае лиц Чернова  $n \leq 22$ . Множество таких наборов можно представлять как точки в пространстве  $R^n$ . Состояние системы мы будем представлять графически в виде рисунка лица, которое состоит из двух симметричных вертикальных половин. Отдельные параметры имеют следующие значения: 1) ширина уха, 2) уровень присоединения уха, 3) половина высоты лица, 4) центр эллипса верхней половины лица, 5) центр эллипса нижней половины лица, 6) длина носа, 7) положение центра рта, 8) кривизна рта, 9) длина рта, 10) высота центра глаза, 11) наклон глаза, 12) центр эллипса глаза, 13) длина глаза, 14) позиция зрачка, 15) место брови (высота), 16) наклон брови, 17) длина брови, 18) радиус уха, 19) ширина носа, 20) наклон волос, 21) длина бороды, 22) густота волос.

Был еще параметр "половина расстояния между глазами", но этот параметр выводится из остальных, и мы его опустим. В Интернете также встречается замена первого параметра на "ширину головы".

Подход прост: для каждой точки  $P \in R^n$  строится изображение  $ChF(P)$ . Параметры назначаются так, чтобы для точек, находящихся в допустимой области работы устройства, лица выглядели "естественно", а для точек вне этой области – вид лица должен бросаться в глаза. Например, рот вылез из лица, см. рис. 7.2.

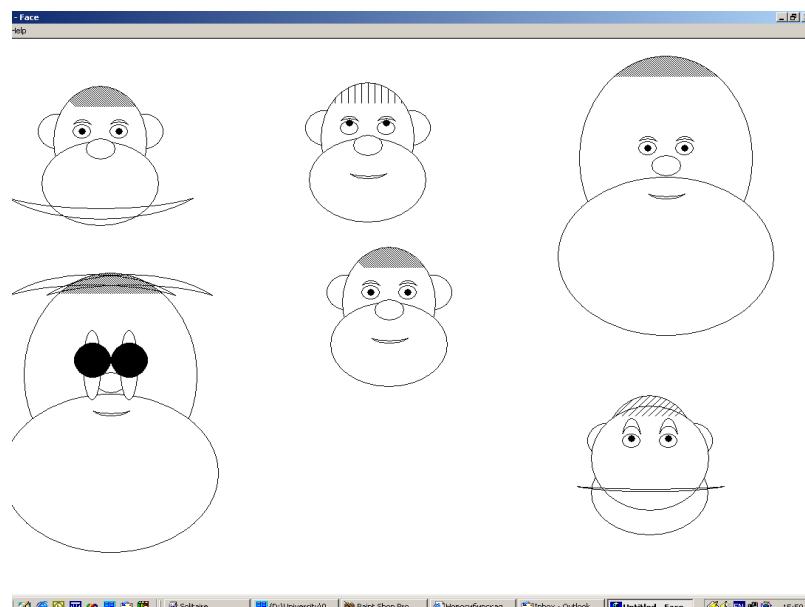


Рис. 7.2. Примеры лиц Чернова

На рис. 7.2 центральные лица изображают нормальный режим работы станции, а остальные показывают аварийные значения каких-либо из контролируемых параметров. Очевидно, что, используя лица Чернова, можно визуализировать наборы из меньшего числа параметров, но стоит иметь в виду, что на этапе установления соответствия параметров и их диапазонов значений с характеристиками лица разработчику придется проделывать огромную работу.

Также см. <http://kspark.kaist.ac.kr/Human%20Engineering.files/Chernoff%20Faces.htm>. Здесь предложена более простая модель – рис. 7.3, лица для крайних допустимых значений параметров приведены на рис. 7.4.

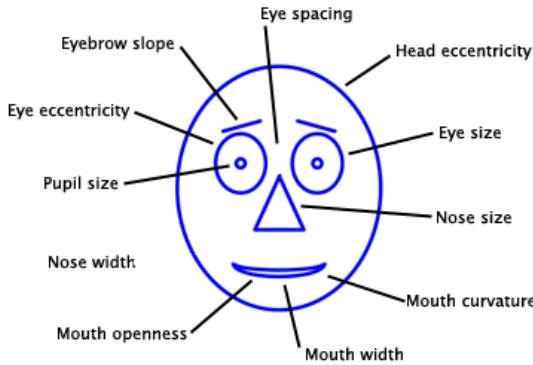


Рис. 7.3. Упрощенная модель лиц Чернова

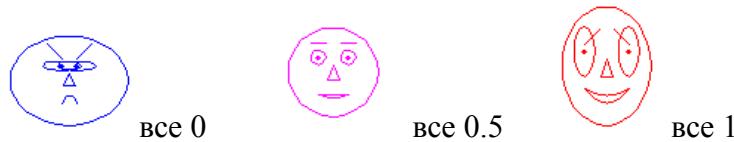


Рис. 7.4. Примеры лиц Чернова по второй модели

В заключение заметим, что данный метод продолжал развиваться и в направлении построения несимметричных лиц [14].

### 7.3.2. Звездчатые графики

Еще один очень часто встречающийся метод графического представления многопараметрических данных – это звездчатые графики, пример которых показан на рис. 7.5.

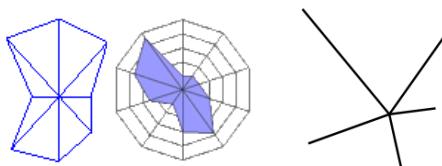


Рис. 7.5. Пример звездчатых графиков

Строятся они следующим образом:

- 1) из точки центра выпускается  $n$  лучей – осей, каждая из которых является осью соответствующего параметра;
- 2) значение параметра из данного набора  $P$  отмечается точкой на соответствующей оси;
- 3) отмеченные точки последовательно соединяются, образуя многоугольник, который и является портретом данного набора.

На рис. 7.5. справа показан другой пример изображения, когда по оси просто откладывается и рисуется отрезок соответствующей длины.

Здесь можно было бы вспомнить и о номограммах, но мы отсылаем читателя к самостоятельному ознакомлению с ними к [16, 17].

### 7.3.3. Кривые Эндрюса как пример функциональных представлений

Среди множества путей представления многопараметрических данных в дву- или трехмерном пространстве графики Эндрюса (Andrew's plots) применяются давно и являются одной из наиболее известных методик [16, 17]. Для набора параметров  $P = \langle p_1, p_2, p_3, \dots, p_n \rangle$  строится портрет в виде графика функции:

$$f_p(t) = \frac{p_1}{\sqrt{2}} + p_2 \sin(t) + p_3 \cos(t) + p_4 \sin(2t) + p_5 \cos(2t) + \dots, -\pi \leq t \leq \pi.$$

Эти функции или кривые Эндрюса обладают свойством, что если два набора или точки  $P_1$  и  $P_2$  лежат близко в пространстве данных  $R^n$ , то и соответствующие им кривые близки визуально.

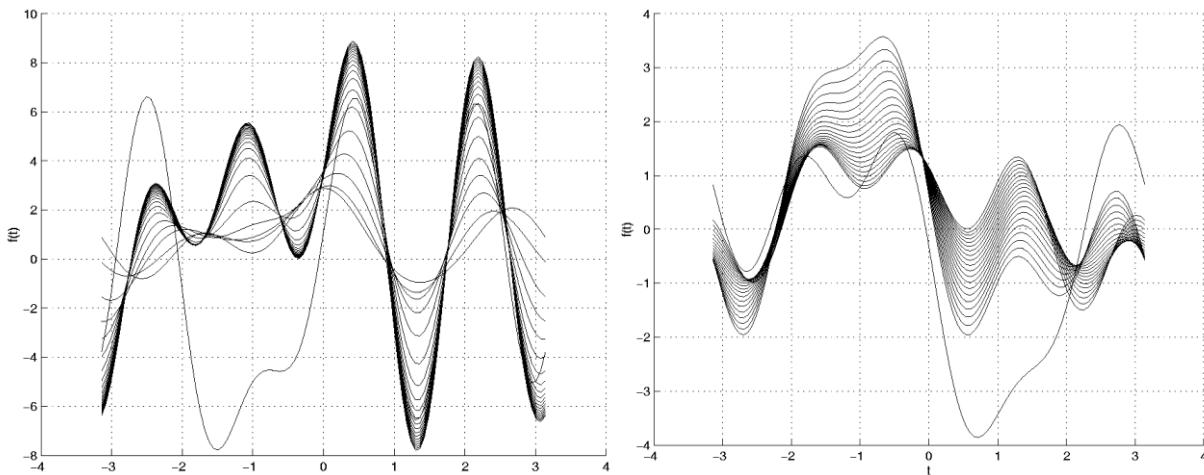


Рис. 7.6. Примеры кривых Эндрюса

На рис. 7.6 изображены совместно графики для нескольких наборов параметров. На обоих рисунках видно, что один из наборов (одна кривая) существенно отличается от остальных, а следовательно, будет в первую очередь интересовать исследователя.

### 7.4. Преобразование координат: вещественные $\Leftrightarrow$ растр

Математическая (или модельная) плоскость, на которой в основном рассчитываются изображения, состоит из точек с вещественными координатами. Для отображения этого изображения используются экран, лазерный принтер, графопостроитель и т.п. Все эти устройства имеют матричную структуру, т.е. их картинная плоскость состоит либо из одинаковых пикселей, либо пишущий элемент может позиционироваться только в узлах некоторой равномерной сетки. Существовали когда-то аналоговые графопостроители, но они не получили распространения в связи с большой погрешностью и развитием техники шаговых двигателей.

Для того чтобы рисунок, рассчитанный на математической плоскости, отобразить на растровом носителе, достаточно преобразовать вещественные координаты ряда основных точек в растровые координаты, поскольку все кривые приближаются семействами отрезков, а отрезок полностью определяется своими концевыми точками.

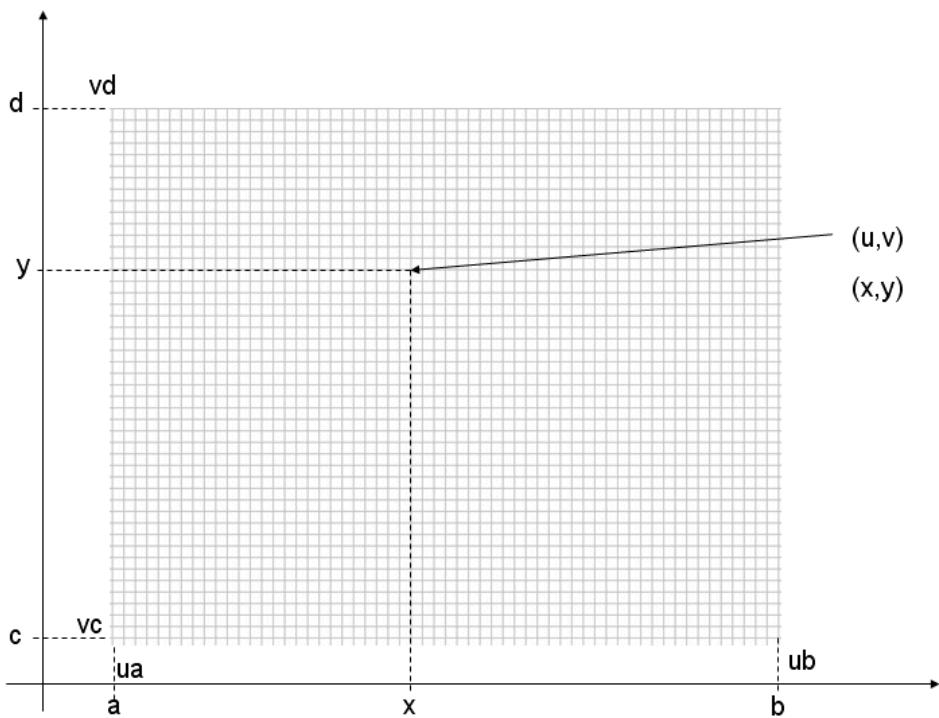


Рис. 7.7. К переводу координат из модельных в растровые

На рис. 7.7. изображена прямоугольная область математической плоскости  $D = [a, b] \times [c, d]$  на растровом прямоугольнике  $R = [ua, ub] \times [vc, vd]$ , узлы сетки рис. 7.7 соответствуют центрам пикселей. Соответствие выбрано таким образом, что точке  $(a, c)$  будет соответствовать центр пикселя  $(ua, vc)$ , точке  $(b, d)$  – центр пикселя  $(ub, vd)$ . Очевидно, что оси модельной и растровой систем координат одинаково ориентированы. Для того чтобы определить центр  $(u, v)$  ближайшего пикселя для образа точки  $(x, y)$  модельной плоскости, достаточно применить следующие формулы:

$$u = \left\lceil \frac{(ub - ua)}{b - a} (x - a) + 0.5 \right\rceil + ua \text{ и } v = \left\lceil \frac{(vd - vc)}{d - c} (y - c) + 0.5 \right\rceil + vb,$$

здесь квадратные скобки означают взятие целой части. Соответственно, обратное преобразование выполняется по формулам:

$$x = \frac{b - a}{ub - ua} (u - ua) + a \text{ и } y = \frac{d - c}{vd - vc} (v - vc) + c.$$

**Замечание.** Необходимо учитывать неточность вещественной арифметики и после перевода из растровых координат в вещественные модельные нужно проверять, что  $a \leq x \leq b$  и  $c \leq y \leq d$ .

Отличительной чертой научного графика являются оси и, особенно, числовые подписи рядом с ними. Здесь можно отметить несколько моментов, неверный учет которых, может привести к построению неверного графика или к неверной интерактивной работе с таким графиком. Мы уже отмечали, что не все вещественные числа представимы в компьютере, еще беднее адресация точек графических устройств. Поскольку мы договорились адресовать пиксели по их центральным точкам, то по горизонтали и вертикали мы имеем дискретные наборы чисел модельной числовой оси, которые реально отображаются в центры пикселей. А главное, это те числа, которые мы можем ввести при помощи указания пикселя на экране. **Еще раз:** с экрана мы не можем ввести произвольное число, только в виде текста. Это замечание необходимо иметь в виду при определении числовой метки для конкретного

пикселя изображенной координатной оси. Поясним на примере. Пусть ось  $X$  размечена подписями  $\{0, 10, 20, 30, 40, 50\}$ , и абсцисса " $x = 30$ " на растре имеет пиксельную абсциссу " $i$ ". Указав мышью на пиксель с абсциссой " $i$ " и выполнив преобразование в математические координаты, мы не можем быть уверены, что получили абсциссу " $x = 30$ ", поскольку точность: полпикселя.

## 7.5. Изолинии, алгоритм марширующих кубиков

Для графического представления однозначных функций двух переменных  $z = f(x, y)$ , заданной на области  $D \subset R^2$ , наиболее часто применяются изолинии или линии уровня. Изолиния является решением уравнения  $f(x, y) = const$ . Вообще-то решением этого уравнения могут быть несколько линий или оно может быть множеством точек не являющимся линией.

Изложим основную идею достаточно древнего, но рабочего, *алгоритма марширующих кубиков* (AMK, marching cubes), см. также [18, 19]. Алгоритм разработан для сеточных функций, т.е.,  $D$  – это прямоугольник –  $D = [a, b] \times [c, d]$ . На этой области определена прямоугольная (в общем случае неравномерная) сетка:

$$a = x_0 < x_1 < x_2 < \dots < x_n = b,$$

$$c = y_0 < y_1 < y_2 < \dots < y_m = d.$$

И даны значения функции в узлах сетки:  $z_{i,j} = f(x_i, y_j), i = 0, \dots, n; j = 0, \dots, m$ .

Выбор сетки осуществляется в прикладной программе и не относится к задаче компьютерной графики. Таким образом, рассматриваемая задача компьютерной графики – это расчет и построение изолиний сеточной функции. Основное предположение АМК – исходная функция  $f$  непрерывна и ведет себя линейно на ребрах сетки.

Сетка разбивает область  $D$  на непересекающиеся прямоугольники меньшего размера – клетки. Изолиния строится на каждом из малых прямоугольников независимо, в результате получаются звенья полной изолинии, которые являются отрезками прямых. Рассмотрим задачу  $f(x, y) = c$  на прямоугольнике  $[x_i, x_{i+1}] \times [y_j, y_{j+1}]$ , см. рис. 7.8.

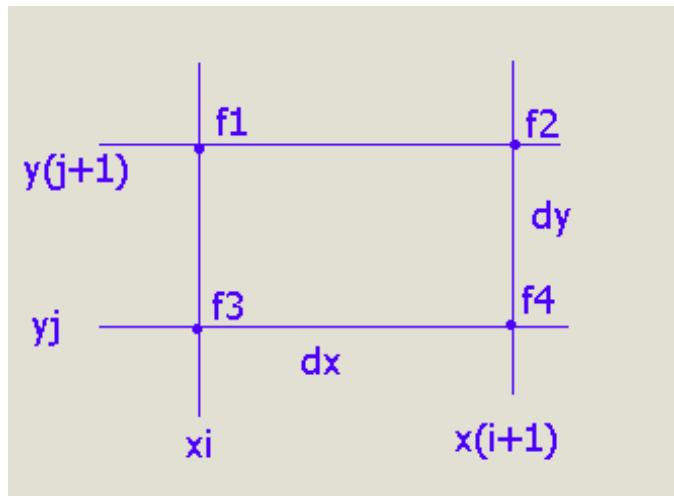


Рис. 7.8. К построению звена изолинии:  $f_1, f_2, f_3, f_4$  – значения функции в узлах

Необходимо найти точки "входа/выхода" изолинии на границах клетки (если таковые есть), которые мы будем называть особыми точками. Так как все 4 стороны рассматриваются независимо, то проведем поиск для верхней границы ячейки. Поскольку считается, что неизвестная функция ведет себя между узлами линейно, изолиния уровня  $c$  не пересекает

верхнюю сторону клетки, если  $(c < f_1) \& (c < f_2)$  или  $(c > f_1) \& (c > f_2)$ . Иначе пересечение существует, т.е. особая точка есть, найдем ее, приняв для определенности, что  $f_1 < f_2$ .

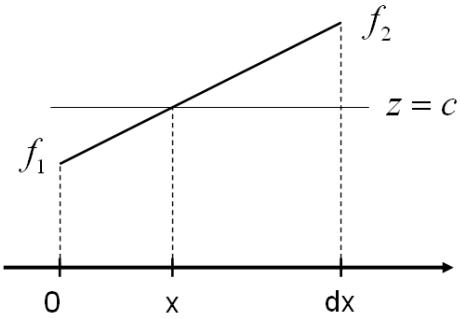


Рис. 7.9. Определение особой точки входа/выхода изолинии на ребре клетки

Согласно рис. 7.9 получаем  $x = dx \frac{c - f_1}{f_2 - f_1}$  в пределах клетки, а на области определения  $D$

функции особая точка входа изолинии в ячейку – это точка

$$\left( x_i + (x_{i+1} - x_i) \frac{c - z_{i,j+1}}{z_{i+1,j+1} - z_{i,j+1}}, y_{j+1} \right).$$

Аналогично определяются и точки на других границах клетки. Рассмотрим возможные ситуации.

- Ни одной точки пересечения изолинии с границей клетки. Это значит, что изолиния не проходит по данной клетке.
- Ровно две особые точки, т.е. найдены точки входа на 2-х из 4-х сторон. Соединяем эти точки отрезком прямой линии.
- 4 точки, т.е. на каждой стороне имеется точка входа. Какие из этих точек соединять? Какие точки соединять с какими, чтобы полная изолиния не имела самопересечений? Этим мы и займемся ниже.
- Упражнение. Может быть и 3 точки. А это как получается?
- Возможны и другие случаи, но мы их рассматривать не будем. Например,  $f_1 = f_2 = f_3 = f_4$  – случай редкий, но возможный. В данной клетке у нас не линия, а целая седловая поверхность в пространстве.

Наиболее важные случаи – это 2, 3 или 4 точки пересечения изолинии с границей клетки. Чтобы определить какие точки соединять с какими в последнем случае можно рассмотреть поведение изолинии в соседних клетках. Или посмотреть, как себя ведут изолинии уровней  $c + \varepsilon$  и  $c - \varepsilon$ . Оказывается и эти приемы могут не дать ответа. Поскольку мы доопределили значение функции на ребрах сетки, но не доопределили для всей области  $D$ . Для иллюстрации приведем пример на рис. 7.10. Оба варианта соединения разумны.

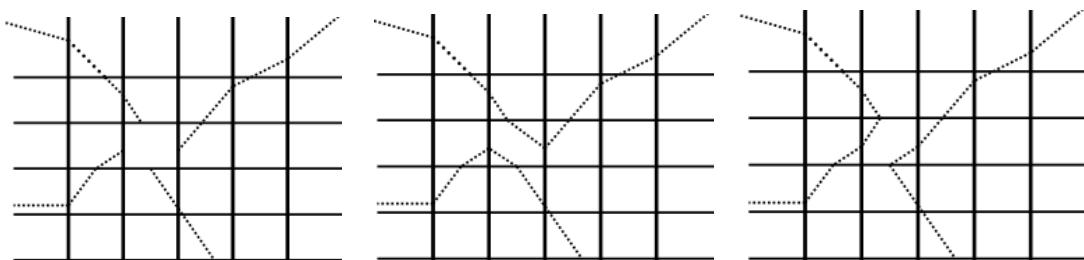


Рис. 7.10. Случай 4-х точек пересечения клетки с изолинией

Большего нам исходная информация дать не может. Пользователь или приложение должны поменять исходные данные, например, задать функцию на более мелкой сетке. Тем не менее, хотелось бы получать однозначное изображение, которое зависит только от исходных данных. Функция  $z = f(x, y)$  однозначная и, следовательно, может быть представлена некоторым рельефом. Задача  $f(x, y) = c$  – это сечение рельефа горизонтальной плоскостью. Теперь нам надо по сеточной функции построить однозначный рельеф. Один из возможных вариантов: представить поверхность рельефа в виде набора пространственных треугольников, для чего вместо прямоугольной сетки рассмотрим сетку из треугольников, например, такую, как показано на рис. 7.11. Известно, что пересечение пространственного треугольника с горизонтальной плоскостью: а) пусто; б) отрезок; в) весь треугольник.

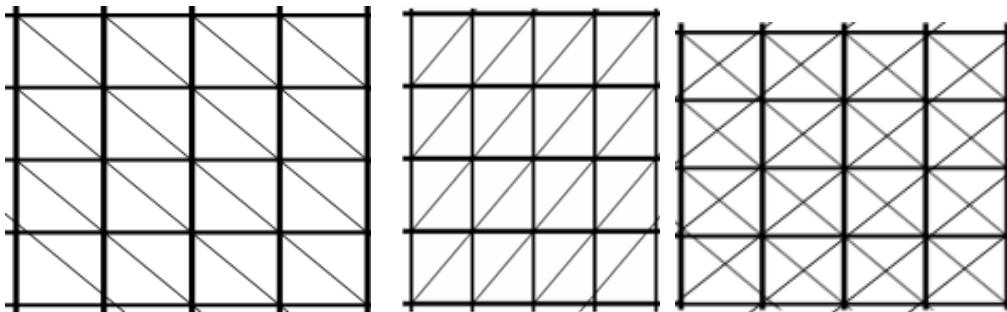


Рис. 7.11. Варианты треугольных сеток

Такое представление рельефа, как слева и посередине, опять непрактично, поскольку простой заменой координатных осей ( $X$  на  $Y$  и наоборот) мы получим другой рельеф – не просто отражение относительно диагонали области. И рисунок изолинии не будет просто отражением рисунка, полученного до замены осей.

#### **Вопрос: Объясните: почему?**

Вариант триангуляции, представленный справа на рис. 7.11, оказывается наиболее устойчивым к упомянутым вольностям при назначении осей в задании сеточной функции. Таким образом, предлагается каждую клетку заменить на 4 треугольника, а для дополнительной средней вершины приписать значение

$$\frac{f_1 + f_2 + f_3 + f_4}{4}.$$

Алгоритм марширующих кубиков достаточно просто распространяется на трехмерный случай, когда решается задача построения изоповерхностей. Аналогично двумерному случаю рассматривается следующая постановка: в узлах прямоугольной сетки

$$a = x_0 < x_1 < x_2 < \dots < x_n = b,$$

$$c = y_0 < y_1 < y_2 < \dots < y_m = d,$$

$$g = z_0 < z_1 < z_2 < \dots < z_l = h,$$

заданы значения функции:  $f_{i,j,k} = f(x_i, y_j, z_k), i = 0, \dots, n; j = 0, \dots, m; k = 0, \dots, l$ . Требуется построить изоповерхности  $f(x, y, z) = const$ .

Применяя алгоритм марширующих кубиков, рассматриваем каждую трехмерную клетку отдельно и, в зависимости от количества особых точек на ребрах клетки, получаем участки изоповерхности в виде пространственных треугольников и четырехугольников. Полученное таким образом множество пространственных многоугольников изображается при помощи стандартных средств трехмерной графики. Для того чтобы уменьшить число рассматриваемых случаев, можно каждую клетку разбить на тетраэдры аналогично двумерному случаю.

*Общее замечание.* В качестве входных данных рассматривается сеточная функция. И хоть в тексте не отмечено специально, при работе алгоритма подразумевается определенное линейное восполнение ее до всей области.

*Цветотоновая карта.* Данный тип графических представлений предназначен для функций, заданных на непрерывной области  $D \subset R^2$ . Карта может изображаться как совместно с изолиниями, так и без них. Применяется следующий алгоритм построения.

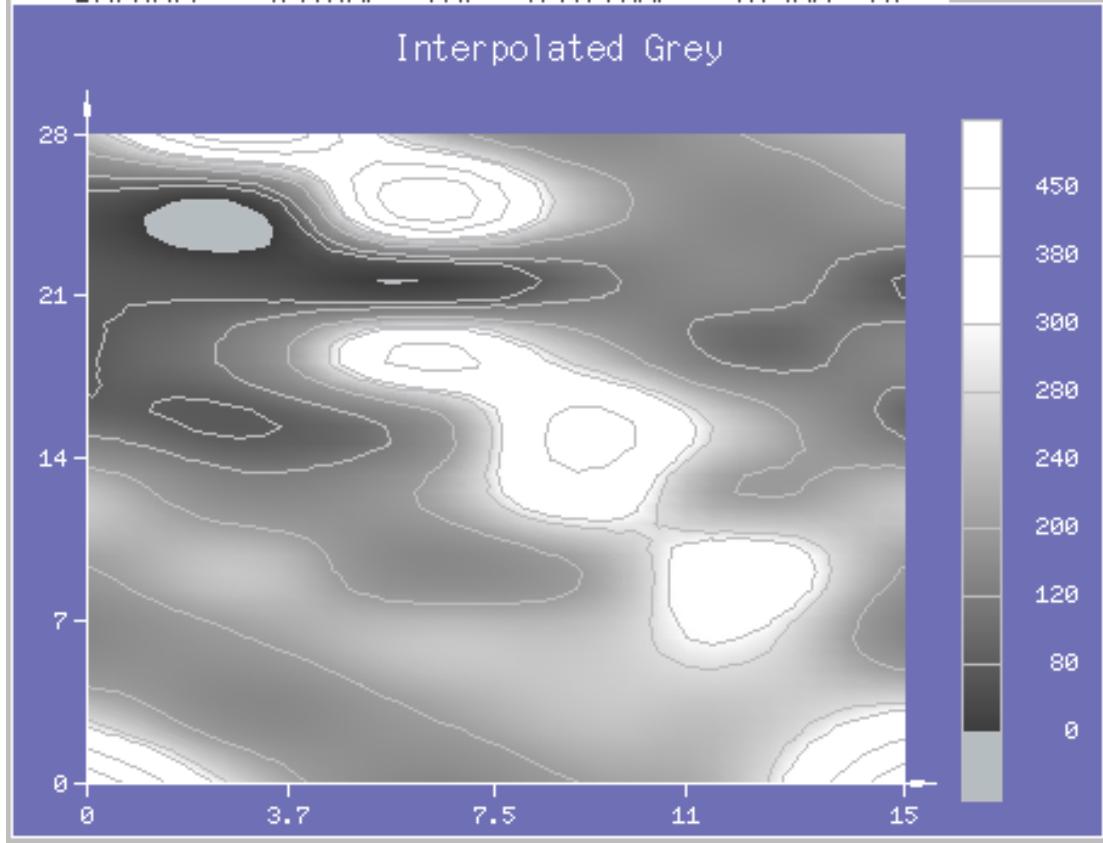


Рис. 7.12. Цветотоновая карта функции. Справа легенда  
соответствия цветов значениям функции

Пусть функция  $z = f(x, y)$  задана на прямоугольнике  $D = [a, b] \times [c, d]$ . Вводится несколько значений – уровней:  $z_1, z_2, \dots, z_t$ ; и набор цветов:  $c_0, c_1, c_2, \dots, c_t$ . Для закрашивания пикселя соответствующего растрового прямоугольника экрана по координатам его центра находим соответствующую точку  $(x, y) \in D$ , и вычисляем в ней значение функции  $z$ . Далее делаем проверки:

- Если  $z < z_1$ , то искомый цвет  $col = c_0$ .
- Если  $z \geq z_t$ , то  $col = c_t$ .
- Во всех остальных случаях ищем индекс  $i$ , такой что  $z_i \leq z < z_{i+1}$  и присваиваем  $col = c_i$ .
- Наконец, выбранный пиксель закрашивается цветом  $col$ .

Полученная по такому алгоритму карта будет представлять набор пиксельных областей постоянного цвета. На рис. 7.12 на цветотоновую карту наложена карта изолиний для уровней  $z_1, z_2, \dots, z_t$ . Области между изолиниями должны иметь пиксели одного цвета, а тут мы видим плавные цветовые переходы. Это связано с тем, что мы применили интерполяцию

цвета, суть которой в следующем: для интервала значений  $z_i \leq z < z_{i+1}$  мы рассматриваем интервал цветов  $c_i \leq col < c_{i+1}$ . Применяем линейную интерполяцию

$$col = (z - z_i) \frac{c_{i+1} - c_i}{z_{i+1} - z_i}$$

по каждой из трех компонент цвета (красный, зеленый, синий). Аналогично интерполируется и легенда.

Напомним еще два способа графического представления карт изолиний – это: применение бергштрихов и числовых меток непосредственно на самих линиях, рис. 7.13 и 7.14.

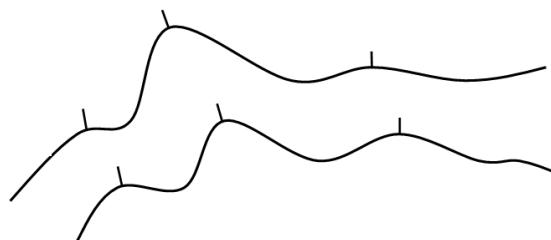


Рис. 7.13. Бергштрихи

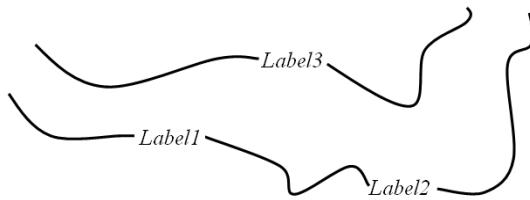


Рис. 7.14. Подписи на изолиниях

*Бергштрих* – это отрезок, нарисованный перпендикулярно к изолинии и указывающий на направление роста значения исследуемой функции. Подписи – это либо номера изолиний, либо числовые значения исследуемой функции. Проблема подписей на изолиниях является частным случаем упоминавшейся проблемы map labeling, кажется простой на первый взгляд. Но это не так, поскольку все распространенные современные программные средства для идентификации линий на научных графиках предпочитают использовать легенды, т.е. карта изолиний отдельно от идентифицирующего списка.

## 7.6. Векторные поля

Достаточно много прикладных исследований в области атмосферы и океана, где важную роль играют различные течения. Для их изучения требуется анализировать скорости в том или ином районе, в определенной точке. В двумерном случае скорость в каждой точке показывается при помощи вектора, изображаемого стрелкой [20].

**Итак, задача:** в узлах прямоугольной сетки

$$a = x_0 < x_1 < x_2 < \dots < x_n = b,$$

$$c = y_0 < y_1 < y_2 < \dots < y_m = d.$$

заданы значения:  $\begin{cases} f_{i,j} = f(x_i, y_j) \\ g_{i,j} = g(x_i, y_j) \end{cases}, i = 0, \dots, n; j = 0, \dots, m$ , определяющие горизонтальную и вертикальную составляющие скорости в узле сетки. Отметим, что наиболее часто применяются сетки равномерные по осям.

**Требуется** построить векторное поле, как на рис. 7.13.

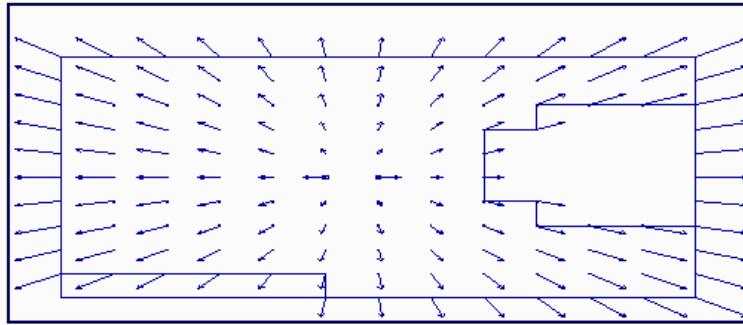


Рис. 7.13. Пример векторного поля

Для того чтобы рисунок векторного поля был достаточно разборчивым – чтобы вектора не создавали картины хаотически разбросанных стрелок – применяются алгоритмы автоматического расчета картины. Рассматриваем случай сетки равномерной по осям. Пусть  $d$  – это диагональ клетки,  $dx$  – шаг сетки по оси  $X$ ,  $dy$  – шаг сетки по оси  $Y$ , а  $L_{i,j}$  – длина вектора (стрелки) в узле  $(i, j)$ .

*Нормировка векторов.* Дополнительная нормировка векторов – коэффициент  $c_l$ . Он выбирается автоматически таким, чтобы вектор  $\begin{pmatrix} c_l \cdot f_{i,j} \\ c_l \cdot g_{i,j} \end{pmatrix}, i = 0, \dots, n; j = 0, \dots, m$ ; не выходил за

пределы одной клетки. Иногда берется более простое условие:  $c_l \cdot L_{i,j} \leq d$ . В ряде случаев пользователь может увеличить или уменьшить значение  $c_l$ , исходя из полученной картинки.

*Шахматный порядок.* Оказывается полезным прием, когда вектора рисуются не во всех узлах, например, узлы выбираются в шахматном порядке.

*Длина стрелки.* Возможен большой разброс значений длин векторов – несколько порядков, тем не менее, имеется желание пользователя видеть направление скорости и оценивать ее величину в сравнении с близлежащими узлами. Можно вектор графически представлять стрелкой, имеющей правильное направление, а длину ее не  $c_l \cdot L_{i,j}$ , а  $\log(1 + c_l \cdot L_{i,j})$ . Один из используемых приемов заключается в следующем. Вводится параметр  $L_{\min}$ , задаваемый в реальных единицах носителя изображения (мм, пиксели). Если требуется рисовать стрелку, длина которой на носителе меньше  $L_{\min}$ , то рисуется специально помеченная стрелка длиной  $L_{\min}$  (стрелка с рисками, см. рис. 7.14). Число рисок – это число порядков, на сколько реальная стрелка короче  $L_{\min}$ .



Рис. 7.14. Маркировка коротких векторов, слева направо: реальная длина, на порядок короче, на два, на три и на четыре порядка короче

*Использование цвета.* В случае построения векторных полей на цветных дисплеях с богатой палитрой и большим диапазоном длин векторов применяется передача величины скорости (длины вектора) цветом. Длина изображения вектора во всех узлах одинакова  $c_l \cdot d$ , а цвет назначен соответственно величине скорости. Для справки рядом с изображением векторного поля дается легенда аналогично цветотоновой карте функций, приведенной на рис. 7.12.

## 8. Визуализация объемных плотностей

Исходное название метода Volume rendering (VR) – визуализация объемов или точнее – визуализация объемных плотностей (ВОП), визуализация среды, взаимодействующей со

светом. Применение метода ВОП/VR – это графический анализ скалярной функции, заданной в пространстве.

Пусть задана скалярная функция  $f(r), r \in D \subset R^3$ . Как правило,  $D$  – это куб. Самый распространенный метод изучения таких функций (по сути четырехмерных геометрических объектов) – это построение изоповерхностей  $\{r \in D : f(r) = \text{const}\}$ . Каждая изоповерхность – это поверхность, составленная из многоугольников. Есть и другие методы графического представления функций, зарекомендовавшие себя, например, см. рис. 8.1, 8.2.

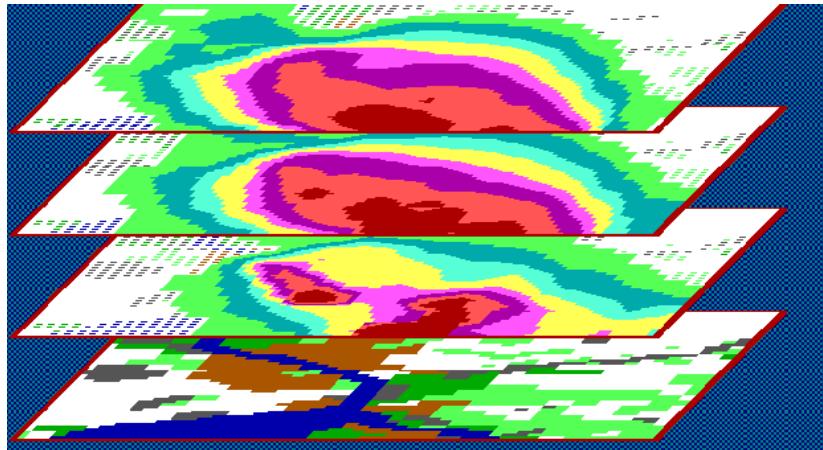


Рис. 8.1. Понижение размерности и изолинии

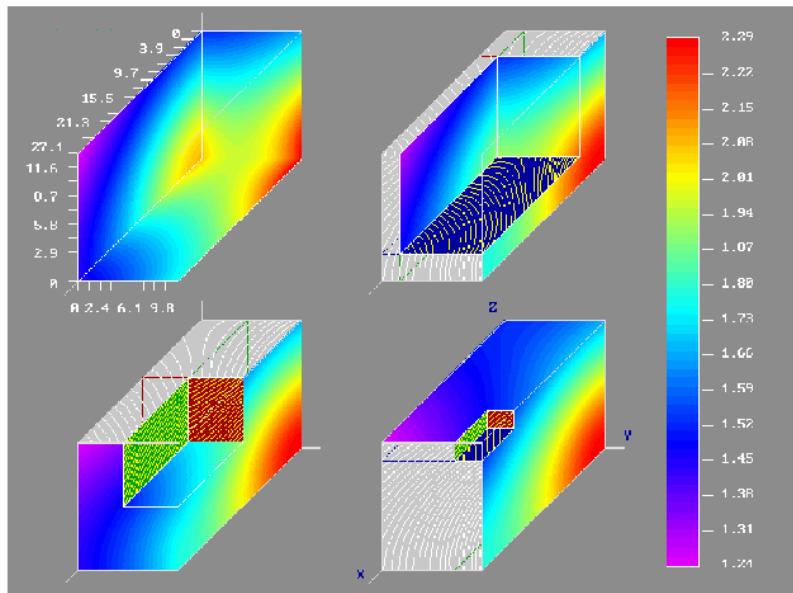


Рис. 8.2. Другое применение изолиний

Как бы мы не получали исходные данные о функции  $f$  (на прямоугольных или хаотических сетках или по аналитическому выражению), мы, тем не менее, считаем, что мы знаем  $f(r)$  в любой точке  $r \in D \subset R^3$ .

Метод ВОП считается лучше, чем методы, описанные выше, поскольку с его помощью исследуется и/или визуализируется сама модель, а не ее граневое представление. Особенно при postprocessing в томографии – врач будет приятно удивлен, увидев непонятные предметы (грани и ребра – не ребра скелета, а ребра 3D модели) в груди пациента.

## 8.1. Основная гипотеза

Полагается, что объемная плотность (ОП) – это "участвующая/реагирующая" среда. Она обладает оптическими свойствами: поглощение, эмиссия, рассеивание, которые влияют на прохождение и/или пропускание луча (свет, Х-лучи, ...). Эта среда состоит, пусть, из капелек воды, из отдельных молекул или чего-то подобного.

Все приведенные ниже выводы касаются только черно-белых лучей. Или для определенной длины волны света. Для цветных образов надо, очевидно, утроить уравнения (для каждого стимула), или применять еще больше уравнений, если рассчитывать по отдельным полосам спектра. Как правило, при помощи цвета исследуется несколько функций одновременно.

Принимается следующая модель ОП:

- Частицы бесконечно малы (шарики).
- Все процессы происходят на любом бесконечно малом отрезке светового луча.

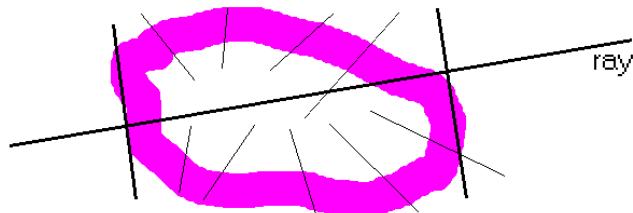


Рис. 8.3. Элементарный объем внутри ОП

В самом общем случае каждый элементарный объем ОП может:

- Поглощать свет – absorption.
- Испускать свет – emission.
- Рассеивать – out scattering.
- Получать свет, "рассеянный" из соседних объемов – in scattering.

Нас интересует исследование ОП при помощи зондирующих лучей. И рассматриваем мы все события относительно тестирующего луча, см. рис. 8.4.

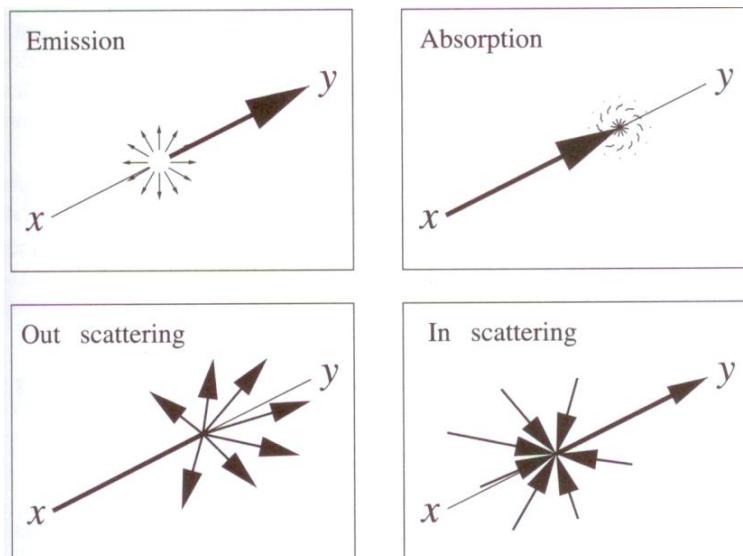


Рис. 8.4.

Проще говоря, луч, проходящий через точку  $x$  в направлении  $y$ , приобретает и теряет энергию по определенным правилам, примеры которых приведены на рис. 8.4. А для каждого элементарного объема можем записать баланс энергии:

$$\Delta W_{out} = \Delta W_{in} + \Delta W_{em} - \Delta W_{abs} - \Delta W_{outSc} + \Delta W_{inSc}.$$

Модели с рассеиванием как очень сложны с математической, так и особенно дороги с вычислительной точки зрения. Поэтому мы остановимся на робастных моделях, в которых учитываются только поглощение и эмиссия. Ведь нам не надо моделировать виртуальную реальность. Наша задача: показать понятно графически функцию.

## 8.1. Только поглощение

Физически среда – это холдные полностью черные частицы, абсорбирующие свет без испускания и рассеивания.

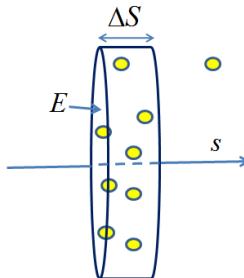


Рис. 8.5. Тонкий слой на пути луча

Представим частицу в виде шарика радиуса  $r$  и проекционной площадью  $A = \pi r^2$ , и пусть:

- $\rho$  – число частиц в единице объема (плотность частиц).
- $E$  – площадь участка (резца), перпендикулярного лучу, рис. 8.5.
- $\Delta S$  – длина участка (толщина слоя) вдоль луча.

Подсчитаем число частиц в рассматриваемом дифференциальном объеме:

$$N = \rho E \Delta S,$$

их проекционная площадь:

$$NA = \rho E \Delta S \pi r^2 = \rho E \Delta S A.$$

Таким образом, часть света, проходящая через рассматриваемый слой, которая будет задета, т.е. попадет в частицу и поглотится, это:  $\rho A \Delta S$ .

Устремим  $\Delta S$  к 0, считая, что вероятность перекрытия проекций частиц равно 0 (очень малое  $\Delta S$ ), получаем дифференциальное уравнение:

$$\frac{dI}{ds} = \rho(s) A I(s) = -\tau(s) I(s), \quad (8.1)$$

здесь  $s$  – параметр длины вдоль луча;  $I(s)$  – интенсивность света в точке луча на расстоянии  $s$ ;  $\tau(s) = \rho(s)A$  – называется коэффициентом ослабления.

Решение уравнения (8.1) записывается в виде:

$$I(s) = I_0 e^{-\int_0^s \tau(t) dt}.$$

$I_0 = I(0)$  – интенсивность луча на входе в объемную плотность.

Величина  $T(s) = e^{-\int_0^s \tau(t) dt}$  – определяет прозрачность среды между 0 (началом среды) и точкой  $s$  на луче. В литературе  $\tau$  часто называют просто *непрозрачностью*. Однако непрозрачность  $\alpha$  для кубика (вокселя) со стороной  $l$ , наблюдаемого в упор, вычисляется как:

$$\alpha = 1 - T(l) = 1 - e^{-\int_0^l \tau(t) dt}.$$

При  $\tau = const$

$$\alpha = 1 - e^{-\tau l} = \tau l - \frac{(\tau l)^2}{2} + \dots$$

Многие пользовательские интерфейсы позволяют специфицировать сразу значение  $\alpha$  для единичной длины  $l$ , таким образом, позволяя задавать  $\tau = \infty$  при  $\alpha = 1$ . Дополнительно предлагается ограничивать непрозрачность сверху единицей, полагая

$$\alpha = \min(1, \tau l).$$

А как же используется визуализируемая функция  $f(x, y, z)$ ? Для этого задается отображение:

$$f \rightarrow \tau,$$

называемое *функцией передачи или пропускания*. Простейшее отображение:

```
if f > K
    τ = ∞
else τ = 0
```

Практически, в случае томографических ОП,  $K$  подбирается так, чтобы кости были непрозрачны, а остальные ткани – прозрачны.

### Визуализация (рендеринг).

1. Визуализация осуществляется с применением алгоритма Z-буфера: чем больше с тем чернее.
2. Для получения гладкой закраски требуется знать нормали:

- a. Используем Z-буфер для оценки наклона проецируемой поверхности;
- b. Либо применяется оценка конечными разностями

$$G(x_i, y_j, z_k) = \begin{pmatrix} \frac{f(x_{i+1}, y_j, z_k) - f(x_{i-1}, y_j, z_k)}{2\Delta x} \\ \frac{f(x_i, y_{j+1}, z_k) - f(x_i, y_{j-1}, z_k)}{2\Delta y} \\ \frac{f(x_i, y_j, z_{k+1}) - f(x_i, y_j, z_{k-1})}{2\Delta z} \end{pmatrix}. \quad \text{При этом}$$

используется и модуль градиента – задаем отображение:  $\tau \sim |G(x, y, z)|$ , которое используется для оценки затухания, позволяя таким образом получать портрет скалярного поля  $f(x, y, z)$  в "Х-лучах", назначая тем или иным образом зависимость  $\tau$  от  $f(x, y, z)$ .

Если считать, что мы смотрим через ОП, то функция  $f(x, y, z)$  играет роль плотности поглощающего облака, рис. 8.6.

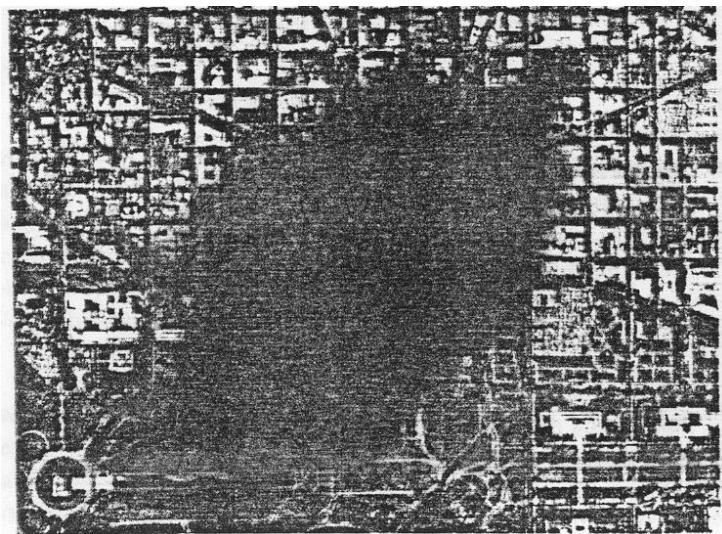


Рис. 8.6.  $f(x, y, z)$  в роли поглощающего облака. Черное облако смога над городом

## 8.2. Только эмиссия

Среда может не только поглощать, но и добавлять свет к тестирующему лучу. Например, частицы пламени. Вернемся снова к рис. 8.5. Пусть частицы прозрачны – не поглощают совсем, но светятся диффузно с интенсивностью  $C$  на единицу проекционной площади. Таким образом, через нашу площадку толщиной  $\Delta s$  ожидаем получить прибавление светового потока  $C\rho A E \Delta s$  или на единицу проекционной площади  $C\rho A \Delta s$ . Тогда при  $\Delta s \rightarrow 0$  получаем дифференциальное уравнение:

$$\frac{dI}{ds} = C(s)\rho(s)A = C(s)\tau(s) = g(s).$$

Ввели  $g(s)$  – компоненту источника.

Решение:

$$I(s) = I_0 + -\int_0^s g(t)dt.$$

Полагая  $g(x, y, z) \sim f(x, y, z)$ , получим изображение похожее на багряное облако, рис. 8.7.



Рис. 8.7. Только эмиссия

### 8.3. Поглощение + эмиссия

Более реалистичное уравнение для облаков учитывает как поглощение света частицами, так и его эмиссию:

$$\frac{dI}{ds} = g(s) - \tau(s)I(s).$$

Будем пока рассматривать независимые функции передачи  $g(f)$  и  $\tau(f)$ . Такая модель очень полезна для визуального анализа непрерывных скалярных полей или медицинских данных, когда  $g$  и  $\tau$  назначаются различным тканям тела.

Решаем. Перенесли часть выражения в левую часть и умножили на  $e^{\int_0^s \tau(t)dt}$ :

$$\left[ \frac{dI}{ds} + \tau(s)I(s) \right] e^{\int_0^s \tau(t)dt} = g(s)e^{\int_0^s \tau(t)dt}.$$

Преобразуем далее:

$$\frac{d}{ds} \left[ I(s)e^{\int_0^s \tau(t)dt} \right] = g(s)e^{\int_0^s \tau(t)dt}.$$

Интегрируем от  $s = 0$  (от начала ОП) до конца ОП  $s = D$  (в месте наблюдателя), и получаем:

$$I(D)e^{\int_0^D \tau(t)dt} - I_0 = \int_0^D g(s)e^{\int_0^s \tau(t)dt} ds.$$

Перенося  $I_0$  в правую часть и умножая на  $e^{-\int_0^D \tau(t)dt}$ , получаем:

$$I(D) = I_0 e^{-\int_0^D \tau(t)dt} + \int_0^D g(s)e^{-\int_s^D \tau(t)dt} ds. \quad (8.2)$$

Первое слагаемое говорит, что начальная интенсивность (на входе в облако) умножается на прозрачность слоя всей ОП. Второе слагаемое – это интеграл, который подсчитывает вклад источника  $g(s)$  для каждого дифференциального участка в точке  $s$  на пути луча сквозь облако с его умножением на прозрачность слоя, оставшегося до конца облака. Прозрачность слоя от  $s$  до  $D$  равна

$$T'(s) = e^{-\int_s^D \tau(t)dt}.$$

Таким образом, окончательно

$$I(D) = I_0 T(D) + \int_0^D g(s) T'(s) ds.$$

### 8.4. Вычисления по модели "поглощение + эмиссия"

Применим простейшую аппроксимацию интеграла в виде суммы Римана:

$$\int_0^D h(x) dx \approx \sum_{i=1}^n h(x_i) \Delta x.$$

Интервал от 0 до  $D$  разобьем на  $n$  равных отрезков длиной  $\Delta x = \frac{D}{n}$ . Выберем на каждом

отрезке  $x_i$  такое, что  $(i-1)\Delta x \leq x_i \leq i\Delta x$ . Для упрощения последующих формул положим:  $x_i = i\Delta x$ . Тогда

$$e^{-\int_s^D \tau(x)dx} \approx e^{-\sum_{i=1}^n \tau(i_{\Delta x})_{\Delta x}} = \prod_{i=1}^n e^{-\tau(i_{\Delta x})_{\Delta x}} = \prod_{i=1}^n t_i,$$

где  $t_i = e^{-\tau(i_{\Delta x})_{\Delta x}}$  может рассматриваться как прозрачность  $i$ -го отрезка на луче и зависит не только от  $\tau(f)$ , но и от длины отрезка  $\Delta x$ . Аналогично проделаем для окончательной формулы (8.2). Положим  $g_i = g(i_{\Delta x})$  и аппроксимируем прозрачность

$$e^{-\int_{i_{\Delta x}}^D \tau(x)dx} \approx \prod_{j=i+1}^n t_j.$$

Также

$$\int_0^D g(s)e^{-\int_s^D \tau(t)dt} ds \approx \sum_{i=1}^n g_i \prod_{j=i+1}^n t_j.$$

Таким образом, получаем окончательное выражение:

$$\begin{aligned} I(D) &= I_0 \prod_{i=1}^n t_i + \sum_{i=1}^n g_i \prod_{j=i+1}^n t_j = \\ &= g_n + t_n(g_{n-1} + t_{n-1}(g_{n-2} + \dots (g_1 + t_1 I_0) \dots)) \end{aligned}$$

Это дает нам известный алгоритм расчета согласно последней формуле:

```
I = I0;
for (I = 1; I <= n; i++)
    I = t[i] * I + g[i];
```

Или другой алгоритм, который позволяет прекратить расчет, как только становится меньше некоторого порога:

```
I = 0;
T = 1 /;
I = n;
While (T > small_threshold && I >= 1)
{
    I = T * I + g[i];
    N = N * t[i];
    i--;
}
I = I + T * I0;
```

Выше мы рассмотрели случай независимого назначения отображений  $g(f)$  и  $\tau(f)$ . Но вспомним, что  $g(s) = C(s)\tau(s)$ . Особенno важен случай, когда цвет эмиссии  $C$  постоянен вдоль луча. Тогда перепишем интеграл из (8.2) более просто:

$$\begin{aligned}
\int_0^D g(s) e^{-\int_s^D \tau(t) dt} ds &= \int_0^D C \tau(s) e^{-\int_s^D \tau(t) dt} ds \\
&= C \int_0^D \frac{d}{ds} e^{-\int_s^D \tau(t) dt} ds \\
&= C \left[ 1 - e^{-\int_0^D \tau(t) dt} \right]
\end{aligned}$$

И окончательно:

$$I(D) = I_0 T(D) + C(1 - T(D)) !$$

Это не что иное, как формула тумана, применяющаяся в библиотеках реалистического рендеринга: OpenGL, DirectX. Цвет фона и цвет тумана комбинируются с учетом прозрачности всего слоя тумана.

## 8.5. Расчет и интерфейс

По мотивам (Paul Bourke, 1996, [http://local.wasp.uwa.edu.au/~pbourke/modelling\\_rendering/](http://local.wasp.uwa.edu.au/~pbourke/modelling_rendering/)).

Рассмотрим задачу ВОП с практической точки зрения. В чем же отличие метода ВОП? В том, что значение  $f(r)$  сопоставляется по некоторому правилу с оптическими свойствами среды в точке  $r \in D \subset R^3$ .

В зависимости от квалификации исследователя он может выбрать наиболее подходящие правила – наиболее подходящую оптическую модель среды. Итак, выбор оптической модели – это дело исследователя и основывается на его прозорливости (таланта), а вот задача компьютерной графики – это обеспечить разнообразие таких моделей и методов их визуализации.

Кроме этого существует и самостоятельная задача машинной графики – это визуализация естественных сред и явлений: туман, дымка и т.д.

Итак, приписывая различные оптические свойства, мы визуально наблюдаем ОП. Заметим, что вместо  $f(r)$  (скалярная функция), мы можем рассматривать  $\bar{f}(r)$ , т.е. векторную функцию или совместное рассмотрение нескольких скалярных функций.

*Исходные данные* – это, как правило, куб (рис. 8.8), разбитый на равные воксели (объемные элементы – кубические ячейки). В каждом вокселе задано всего 1 значение.

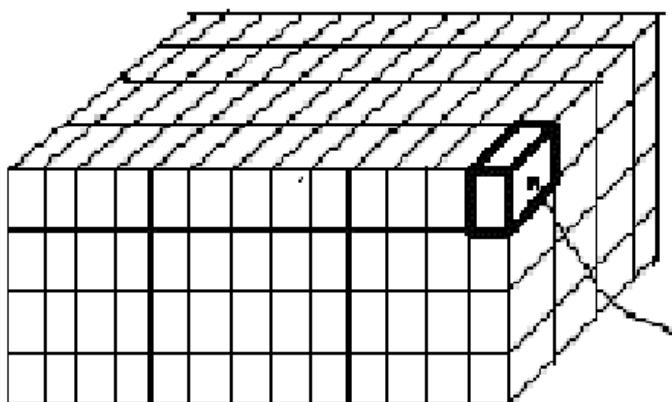


Рис. 8.8. Куб с исходными данными

Каковы объемы исходных данных? Если куб  $100 \times 100 \times 100$ , а значение занимает всего 1 байт, то уже 1МБ, а если  $300 \times 300 \times 300 * 1$  байт = 27 МБ.

На предварительном этапе исходную исследуемую скалярную функцию масштабируют и сдвигают так, чтобы все значения лежали в пределах от 0 до 255 (1 байт). Рассмотрим примерный сценарий работы исследователя на следующем наборе данных – см. рис. 8.9.

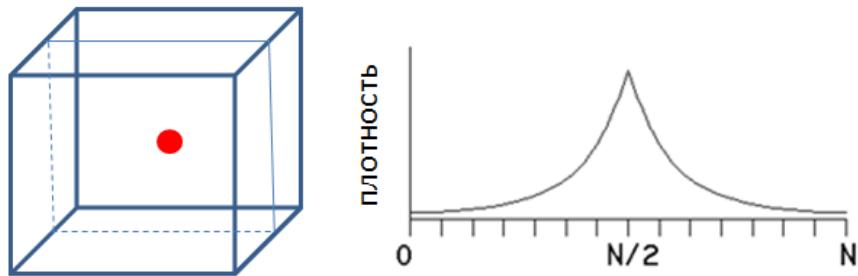


Рис. 8.9. График значений плотности в центральном сечении

Данные имеют центральную симметрию и могут рассматриваться, как поле от точечного заряда.

1. Задаем функцию передачи на основе непрозрачности, рис. 8.10.

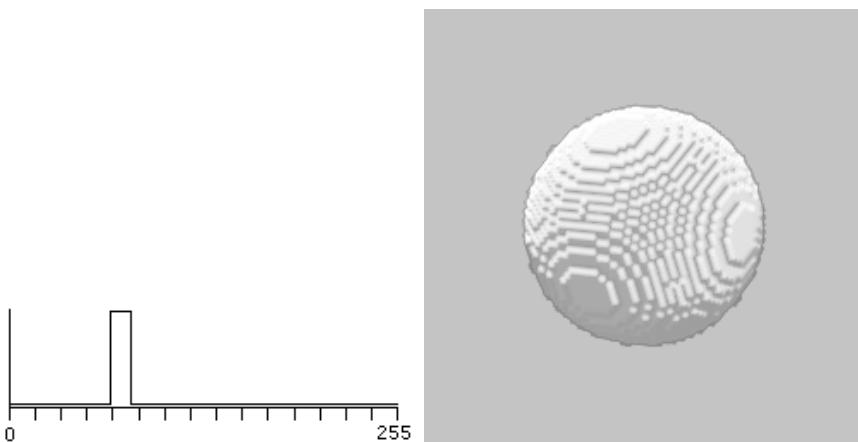


Рис. 8.10. Функция передачи – график непрозрачности. Справа изображение

Здесь мы всем возможным значениям куба от 0 до 255 задаем полную прозрачность (нулевое значение), и только небольшой диапазон значений – полная непрозрачность (=1). Таким образом, мы могли бы пытаться найти определенную ткань тела, например, кости.

2. Потребуем плавную передачу в некотором диапазоне значений – рис. 8.11. Результат – облако.

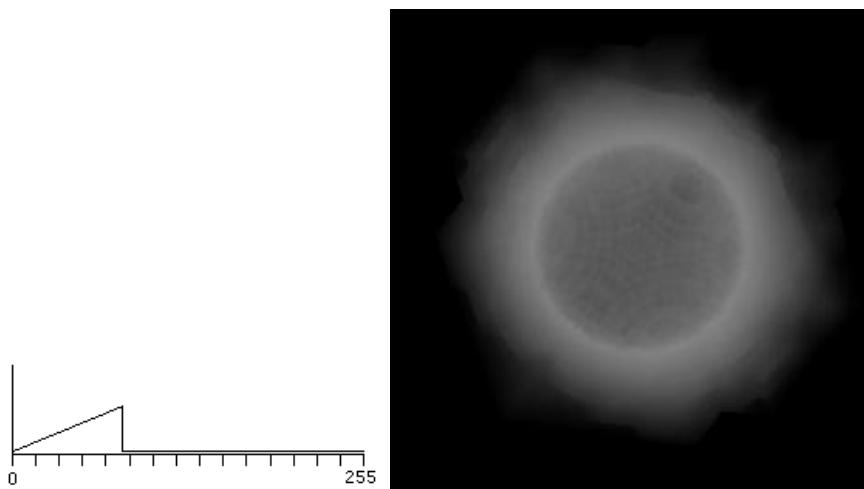


Рис. 8.11. Плавная передача в диапазоне. Получилось облако

3. Комбинируем – рис. 8.13.

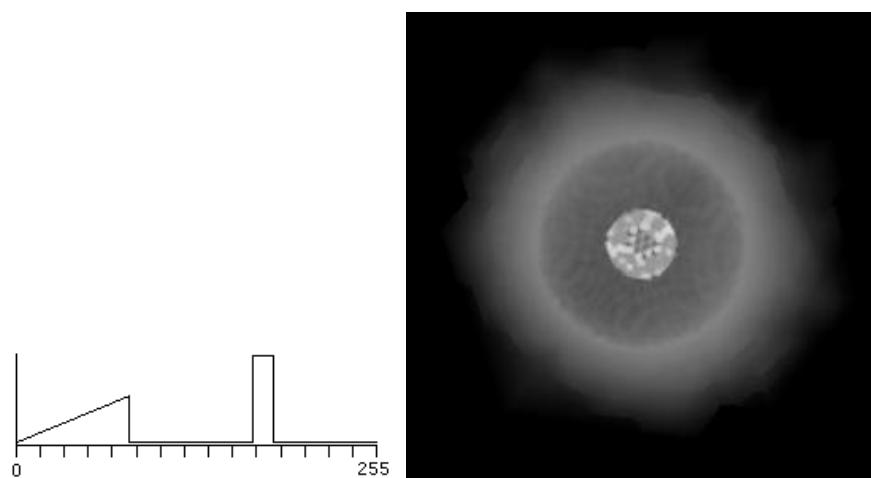


Рис. 8.12. Добавилось облако вокруг прежнего ядра с рис 8.11

4. Подключаем функцию передачи цветом – рис. 8.13.

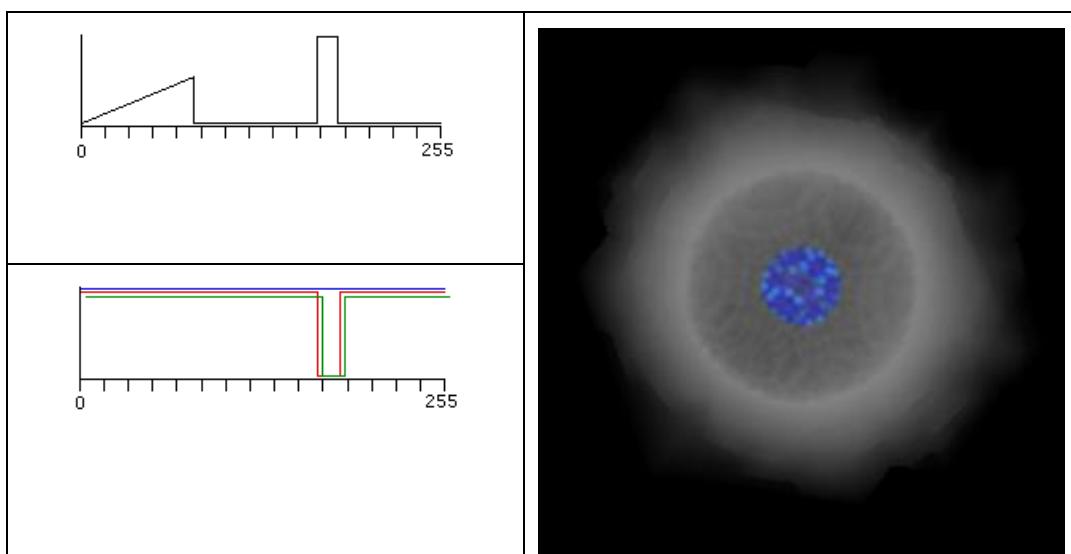


Рис. 8.13. Две функции передачи совместно, ядро стало выделяться из-за цвета

5. Более искусшенное управление цветом – рис. 8.14, результат – рис. 8.17.

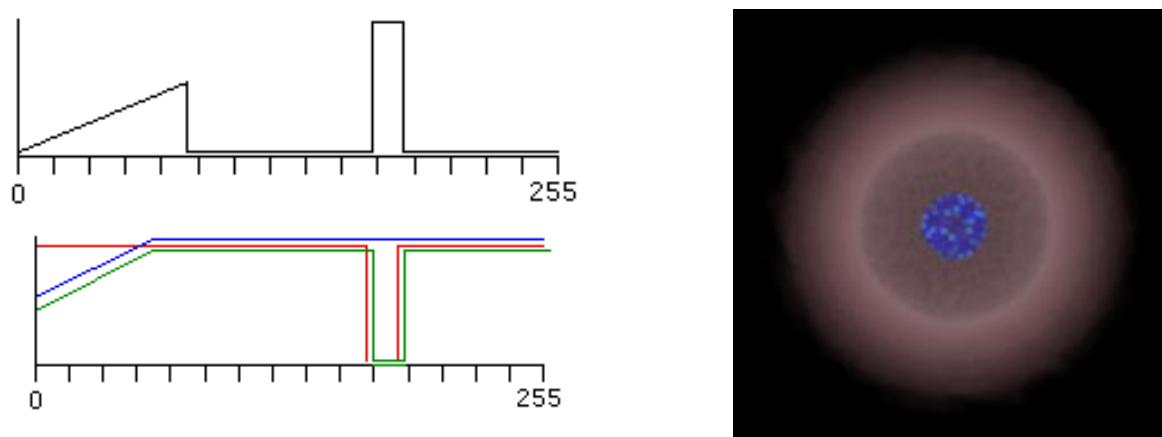


Рис. 8.14. Сложное управление цветом. Окружающее облако порозовело

## 9. Элементы вычислительной геометрии на плоскости и в пространстве

В данном разделе будет кратко изложен материал, хорошо знакомый из курса аналитической геометрии [21] и дифференциальной геометрии [22]. Основная задача – это выработать единую терминологию и обозначения, и, главное, усвоить важные с точки зрения компьютерной графики понятия: ориентация, справа/слева, параметрическое задание и т.п. Объем материала достаточен для изложения данной части курса и для выполнения практических заданий.

### 9.1. Плоскость

Итак, имеется плоскость  $R^2$ , на которой введена декартова система координат.

#### 9.1.1. Точки и векторы

Каждая точка плоскости характеризуется двумя вещественными координатами и

$$P = \begin{pmatrix} p_x \\ p_y \end{pmatrix} \text{ или } Q = \begin{pmatrix} q_x \\ q_y \end{pmatrix}.$$

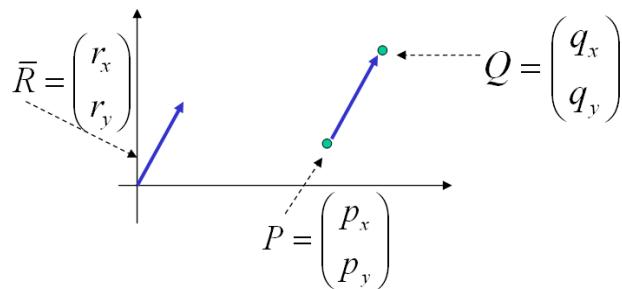


Рис. 9.1. Точки, векторы

Вектор – это направленный отрезок  $\bar{R} = \begin{pmatrix} r_x \\ r_y \end{pmatrix}$ , начинающийся в начале системы координат.

Часто вектора изображают и в других местах плоскости, например, вектор от  $P$  к  $Q$  на рис. 9.1. Это делается для удобства изложения, т.к. при помощи векторов задаются направления. А на самом деле  $\bar{R} = \overrightarrow{PQ} = \begin{pmatrix} q_x - p_x \\ q_y - p_y \end{pmatrix}$ .

Евклидово расстояние между двумя точками плоскости:  $|Q - P| = \sqrt{(q_x - p_x)^2 + (q_y - p_y)^2}$ .

Длина вектора:  $|\bar{R}| = \sqrt{r_x^2 + r_y^2}$ .

#### 9.1.2. Отрезок

Рассмотрим параметрические способы задания отрезка:

1. Двумя концевыми точками  $P$  и  $Q$ . Любая точка отрезка описывается выражением с параметром  $t$ :  $S(P, Q, t) = (1-t)P + tQ, t \in [0, 1]$ .
2. Начальной точкой  $P$ , направляющим вектором  $\bar{u} = \begin{pmatrix} u_x \\ u_y \end{pmatrix}$  и длиной  $a$ . Точка отрезка:

$$S(P, \bar{u}, a, t) = P + t \frac{\bar{u}}{|\bar{u}|}, t \in [0, a].$$

Отметим, что при таких определениях мы можем говорить об *ориентации* отрезка, если будем учитывать значение  $t$ , т.к. нами определены его начало и конец.

Наиболее удобным с точки зрения записи алгоритмов и даже вычислений является случай направляющих векторов единичной длины:  $|\bar{u}|=1$ .

В дальнейшем мы не будем применять отдельные специальные обозначения для вектора, точки или отрезка, если по контексту будет понятно, о чём идет речь.

### 9.1.3. Прямая и луч

Параметрические задания:

1. Прямая, проходящая через точку  $P$  в направлении точки  $Q$ :

$$L(P, Q, t) = (1-t)P + tQ, t \in [-\infty, +\infty]. \text{ Если } t \in [0, +\infty], \text{ то это уравнение луча.}$$

2. Прямая, проходящая через точку  $P$ , с направляющим вектором  $\bar{u}$ :  $L(P, \bar{u}, t) = P + t \cdot \bar{u}, t \in [-\infty, +\infty]$ . Если  $t \in [0, +\infty]$  – луч. Нормаль к прямой или лучу:

$$\bar{n} = \begin{pmatrix} n_x \\ n_y \end{pmatrix} = \begin{pmatrix} -u_y \\ u_x \end{pmatrix}. \text{ Кстати, это верно и для отрезка. Отметим, что данный вектор } \bar{n}$$

получен поворотом направляющего вектора  $\bar{u}$  на  $90^\circ$  против часовой стрелки, так сказать, положительное вращение.

Часто в исходном задании прямая представляется уравнением гиперплоскости:

$$L(a, b, c) :: ax + by + c = 0. \quad (9.1)$$

Если мы знаем какую-нибудь точку  $P$ , принадлежащую этой прямой, то любая точка  $Q$  прямой должна удовлетворять выражению:  $(Q - P, \begin{pmatrix} a \\ b \end{pmatrix}) = 0$ . Нормаль к прямой равна  $\begin{pmatrix} a \\ b \end{pmatrix}$ .

Мы можем также переписать данную прямую в параметрическом виде:

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} \frac{-ac}{a^2+b^2} \\ \frac{b}{a^2+b^2} \end{pmatrix} + t \cdot \begin{pmatrix} b \\ -a \end{pmatrix}, \text{ правда, теперь она уже является ориентированной, и мы можем}$$

говорить такие выражения, как: "точка находится справа/слева от прямой". Например, "слева" – это в полуплоскости, на которую указывает нормаль к данной прямой.

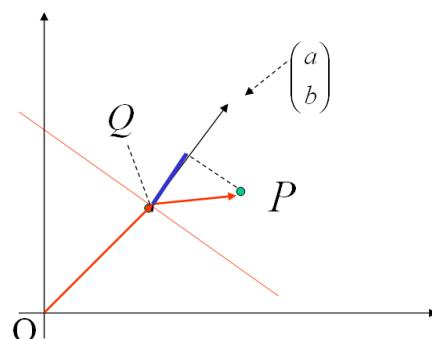


Рис. 9.2. О расстоянии от точки до прямой

Важная величина – это расстояние от точки  $P$  до прямой  $L$ , для задания (9.1) оно определяется по формуле:  $\rho(P, L) = \frac{a \cdot p_x + b \cdot p_y + c}{\sqrt{a^2 + b^2}}$ . Оно имеет знак. Это знак "+", если

точка  $P$  лежит слева от прямой  $L$ , и знак "-", если справа. На рис. 9.2 объясняется предыдущее высказывание, точка  $Q$  – это произвольная точка, лежащая на прямой.

$$\begin{aligned}
\rho(P, L) &= \left( \begin{pmatrix} a \\ b \end{pmatrix}, P \right) + c = (\bar{n}, P) + c = \\
&= (\bar{n}, Q + \overline{QP}) + c = (\bar{n}, Q) + c + (\bar{n}, \overline{QP}) = \\
&= -c + c + (\bar{n}, \overline{QP}) = (\bar{n}, \overline{QP})
\end{aligned}$$

Таким образом, мы показали, что по формуле (9.1) мы также задаем ориентированную прямую.

Очень популярно представление прямой уравнением  $y = mx + b$ . Но оно не нашло большого применения в вычислительной геометрии, поскольку с его помощью невозможно представить вертикальную прямую, например,  $x = 5$ . Все рассмотренные выше представления не имеют такого недостатка.

*Упражнение.* Записать уравнение прямой  $y = mx + b$  в параметрическом виде.

#### 9.1.4. Углы

Углы между отрезками, прямыми и лучами равны углам между их направляющими векторами или нормалями. Через  $(\bar{u}, \bar{v})$  будем обозначать скалярное произведение, через  $\bar{u} \times \bar{v}$  – векторное произведение двух векторов  $\bar{u}$  и  $\bar{v}$ .

Функции угла между двумя векторами  $\bar{u}$  и  $\bar{v}$  вычисляются по следующим формулам:

- Косинус  $\cos(\bar{u}, \bar{v}) = \frac{(\bar{u}, \bar{v})}{|\bar{u}| \cdot |\bar{v}|}$ ;
- Синус  $\sin(\bar{u}, \bar{v}) = \frac{\bar{u} \times \bar{v}}{|\bar{u}| \cdot |\bar{v}|} = \frac{u_x v_y - u_y v_x}{\sqrt{u_x^2 + u_y^2} \cdot \sqrt{v_x^2 + v_y^2}}$ ;
- Вспомним, что  $\sin(\bar{u}, \bar{v}) = -\sin(\bar{v}, \bar{u})$ .

Две прямые  $L(a_1, b_1, c_1)$  и  $L(a_2, b_2, c_2)$  параллельные, если  $a_1 b_2 - a_2 b_1 = 0$ , иначе их точка

$$\begin{pmatrix} \frac{b_1 c_2 - b_2 c_1}{a_1 b_2 - a_2 b_1} \\ \frac{c_1 a_2 - c_2 a_1}{a_1 b_2 - a_2 b_1} \end{pmatrix}.$$

пересечения определяется по формуле:

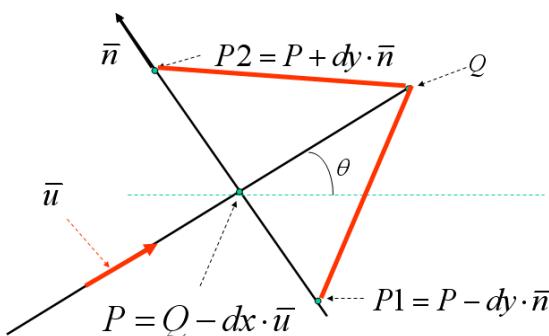


Рис. 9.3. Расчет стрелки

При формировании рисунков достаточно часто возникает необходимость нарисовать стрелку – отрезок с наконечником. Большинство студентов сразу начинают применять

тригонометрию с углом  $\theta$  (см. рис. 9.3) для вычисления этого наконечника, хотя использование параметрического задания отрезка куда экономнее по объему вычислений, да и нагляднее. И практически не бывает ошибок в отличие от тригонометрического случая. На рис. 9.3 показан весь такой расчет:  $\bar{u}$  – единичный направляющий вектор отрезка,  $\bar{n}$  – единичный вектор нормали к отрезку,  $dx$  и  $dy$  – параметры наконечника.

## 9.2. Пространство

Рассмотрим пространство  $R^3$  с декартовой системой координат. Как мы увидим, одна часть объектов и понятий почти не изменилась по сравнению с плоским случаем из-за появления дополнительной третьей координаты, но есть случаи, когда понятие не имеет в пространстве точного определения, например, нормаль к прямой.

### 9.2.1. Точки, векторы, отрезки и прямые

Каждая точка пространства характеризуется тремя вещественными координатами и обозначается  $P = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}$  или  $Q = \begin{pmatrix} q_x \\ q_y \\ q_z \end{pmatrix}$ . Вектор также имеет три координаты  $\bar{R} = \begin{pmatrix} r_x \\ r_y \\ r_z \end{pmatrix}$ . И аналогично плоскому случаю:

- Евклидово расстояние между точками:  $|P - Q| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2 + (p_z - q_z)^2}$ .
- Длина вектора:  $|\bar{R}| = \sqrt{r_x^2 + r_y^2 + r_z^2}$ .
- Вектор как направление от точки к точке:  $\bar{R} = \overrightarrow{PQ} = \begin{pmatrix} q_x - p_x \\ q_y - p_y \\ q_z - p_z \end{pmatrix}$
- Отрезок с концевыми точками  $P$  и  $Q$ . Любая точка отрезка описывается выражением с параметром  $t$ :  $S(P, Q, t) = (1-t)P + tQ$ ,  $t \in [0, 1]$ .
- Отрезок с начальной точкой  $P$ , направляющим вектором  $\bar{u} = \begin{pmatrix} u_x \\ u_y \end{pmatrix}$  и длиной  $a$ . Точка отрезка:  $S(P, \bar{u}, a, t) = P + t \frac{\bar{u}}{|\bar{u}|}$ ,  $t \in [0, a]$ .
- Прямая, проходящая через точку  $P$  в направлении точки  $Q$ :  $L(P, Q, t) = (1-t)P + tQ$ ,  $t \in [-\infty, +\infty]$ . Если  $t \in [0, +\infty]$ , то это уравнение луча.
- Прямая, проходящая через точку  $P$ , с направляющим вектором  $\bar{u}$ :  $L(P, \bar{u}, t) = P + t \cdot \bar{u}$ ,  $t \in [-\infty, +\infty]$ . Если  $t \in [0, +\infty]$  – луч.
- А вот нормаль к прямой в пространстве однозначно не определяется, аналогично для луча и отрезка.

*Упражнение.* Вычислить расстояние от точки  $Q$  до прямой  $P + t \cdot \bar{u}$ .

### 9.2.2. Плоскость в пространстве

Следующее уравнение задает плоскость:

$$\Pi(a, b, c, d) :: ax + by + cz + d = 0. \quad (9.2)$$

Если мы знаем какую-нибудь точку  $P$ , принадлежащую этой плоскости, то любая точка  $Q$

плоскости должна удовлетворять выражению:  $\left(Q - P, \begin{pmatrix} a \\ b \\ c \end{pmatrix}\right) = 0$ . Нормаль к плоскости равна

$\begin{pmatrix} a \\ b \\ c \end{pmatrix}$ . Мы можем говорить об ориентации плоскости и такие выражения, как: "точка

находится справа/слева от плоскости". Например, "слева" – это в полупространстве, на которое указывает нормаль к данной плоскости.

Расстояние от точки  $P$  до плоскости  $\Pi$ , для задания (9.2) определяется по формуле:

$$\rho(P, \Pi) = \frac{a \cdot p_x + b \cdot p_y + c \cdot p_z + d}{\sqrt{a^2 + b^2 + c^2}}. \text{ Оно имеет знак, который позволяет определять по какую}$$

сторону от плоскости лежит точка. Если знак "+", то будем говорить, что точка  $P$  лежит слева от плоскости, и знак "-", если справа.

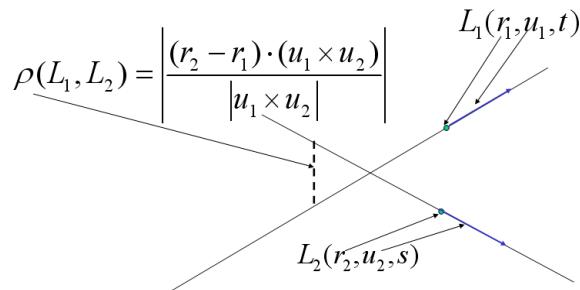


Рис. 9.4. Расстояние между скрещивающимися прямыми

В пространстве возможен случай скрещивающихся прямых, расстояние между которыми вычисляется, как показано на рис. 9.4.

Углы между отрезками, прямыми и лучами равны углам между их направляющими векторами, а углы между плоскостями равны углам между их нормалами. И, в заключение, приведем формулу для вычисления синуса угла между двумя векторами  $\bar{u}$  и  $\bar{v}$ :

$$\sin(\bar{u}, \bar{v}) = \frac{\bar{u} \times \bar{v}}{|\bar{u}| \cdot |\bar{v}|} = \frac{\begin{vmatrix} i & j & k \\ u_x & u_y & u_z \\ v_x & v_y & v_z \end{vmatrix}}{|\bar{u}| \cdot |\bar{v}|}.$$

### 9.3. Симплексы и барицентрические координаты

Симплекс – это простейшая фигура, имеющая ненулевую меру в данном пространстве [9]. В зависимости от размерности различаются следующие симплексы:

*точка* – симплекс размерности 0;

*отрезок* – симплекс размерности 1, 2 опорные точки;

*треугольник* – симплекс размерности 2, 3 опорные точки;

*тетраэдр* – симплекс размерности 3, 4 опорные точки.

Все они жесткие выпуклые фигуры. Для последних трех симплексов имеют важное значение, так называемые, *барицентрические координаты*. Пусть  $P_1, \dots, P_n$  – это опорные точки

(вершины), тогда любая точка симплекса  $P$  может быть представлена в виде следующей суммы:

$$\lambda_1 P_1 + \dots + \lambda_n P_n, \text{ при этом } \lambda_1 + \dots + \lambda_n = 1 \text{ и } \lambda_i \geq 0, i = 1, \dots, n.$$

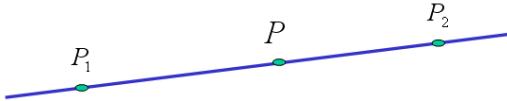


Рис. 9.5. Отрезок

Действительно, отрезок может быть представлен в виде:  $(1-t)P_1 + tP_2$ ,  $0 \leq t \leq 1$ . И наоборот,

любая точка  $P$  отрезка  $\overrightarrow{P_1P_2}$  может быть представлена как  $\frac{|P_2 - P|}{|P_2 - P_1|}P_1 + \frac{|P - P_1|}{|P_2 - P_1|}P_2$  (рис. 9.5),

т.е все условия выполнены. Отрезок – это симплекс максимальной размерности в пространстве размерности 1, т.е. на прямой. Мы уже отмечали ранее, что прямая у нас ориентированная фигура, и ее ориентация может определяться направляющим вектором  $\bar{u}$ . Пусть одна из точек прямой – точка  $P_0$  – является ее началом. Очевидно, что существуют значения параметра  $t_1$  и  $t_2$ , такие что  $P_1 = P_0 + t_1 \cdot \bar{u}$  и  $P_2 = P_0 + t_2 \cdot \bar{u}$ ,  $|\bar{u}| = 1$ . Тогда длиной отрезка  $\overrightarrow{P_1P_2}$  будем считать  $\mu(\overrightarrow{P_1P_2}) = t_2 - t_1$ . Таким образом определенная длина позволяет нам говорить, что отрезок ориентирован положительно на прямой, если длина больше нуля, и ориентирован отрицательно, если длина меньше нуля. Независимо от ориентации отрезка на прямой барицентрические координаты точек отрезка зависят только от абсолютных расстояний:  $\lambda_1 = \frac{|P_2 - P|}{|P_2 - P_1|}$  и  $\lambda_2 = \frac{|P - P_1|}{|P_2 - P_1|}$ . Более того, любая точка данной прямой может быть представлена в виде  $\lambda_1 P_1 + \lambda_2 P_2$  при  $\lambda_1 + \lambda_2 = 1$ , У точек, не принадлежащих отрезку  $\overrightarrow{P_1P_2}$  будет нарушение одного из условий:  $\lambda_i \geq 0, i = 1, 2$ .

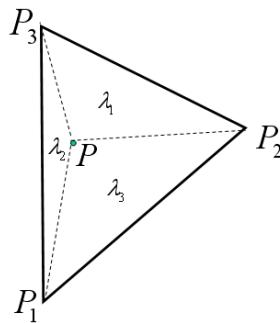


Рис. 9.6. Треугольник

Теперь рассмотрим треугольник. Опорные точки треугольника – это его вершины, рис. 9.6. Тогда любая точка  $P$  треугольника, включая граничные, представляется в виде:  $\lambda_1 P_1 + \lambda_2 P_2 + \lambda_3 P_3$  при  $\lambda_1 + \lambda_2 + \lambda_3 = 1$  и  $\lambda_i \geq 0, i = 1, 2, 3$ . Оказывается, что формулы для определения барицентрических координат также основываются на мере – на площади треугольника. Вспоминаем, что площадь треугольника в  $R^2$ , рис. 9.6, определяется выражением:  $S(P_1P_2P_3) = \frac{1}{2} \cdot \left[ (P_{2x} - P_{1x}) \cdot (P_{3y} - P_{1y}) - (P_{3x} - P_{1x}) \cdot (P_{2y} - P_{1y}) \right]$ .

Получаемые значения площадей могут иметь знак. Договоримся считать треугольник положительно ориентированным, если значение площади положительно, и отрицательно ориентированным – в противном случае. Если треугольник положительно ориентирован, то при обходе его границы согласно нумерации вершин его внутренность будет оставаться слева. Теперь вычисляем значения самих координат:

$$\lambda_1 = \frac{S(PP_2P_3)}{S(P_1P_2P_3)}; \quad \lambda_2 = \frac{S(PPP_3)}{S(P_1P_2P_3)}; \quad \lambda_3 = \frac{S(P_1P_2P)}{S(P_1P_2P_3)}.$$

Очевидно, что  $\lambda_1 + \lambda_2 + \lambda_3 = 1$  и  $\lambda_i \geq 0, i = 1, 2, 3$ . Аналогично случаю отрезка любая точка плоскости данного треугольника может быть представлена в таком же виде. Но если точка этой плоскости не принадлежит треугольнику, то будет нарушено, по крайней мере, одно из условий  $\lambda_i \geq 0, i = 1, 2, 3$ .

Для треугольника в пространстве для нахождения квадрата площади треугольника можно воспользоваться свойством векторного произведения, что длина вектора результата равна

$$\text{удвоенной площади треугольника } S(P_1P_2P_3) = \left| \frac{1}{2} \cdot \begin{vmatrix} i & j & k \\ P_{2x} - P_{1x} & P_{2y} - P_{1y} & P_{2z} - P_{1z} \\ P_{3x} - P_{1x} & P_{3y} - P_{1y} & P_{3z} - P_{1z} \end{vmatrix} \right|. \quad K$$

сожалению, здесь мы потеряли знак, что не позволяет использовать это выражение для вычисления барицентрических координат прямо в пространстве  $R^3$ .

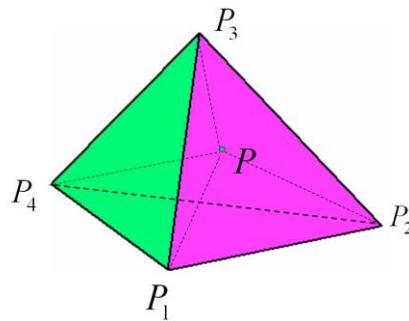


Рис. 9.7. Тетраэдр

И, наконец, рассмотрим тетраэдр, см. рис. 9.7. Объем тетраэдра вычисляется по формуле смешанного произведения векторов-сторон:

$$V(P_1, P_2, P_3, P_4) = \frac{1}{6} \begin{vmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{vmatrix} = \frac{1}{6} ((P_2 - P_1), (P_3 - P_1), (P_4 - P_1)).$$

Достаточно поменять нумерацию двух вершин тетраэдра, как значение объема поменяет знак. Это не сказывается на определении барицентрических координат, но позволяет с тетраэдром также связать понятие положительной и отрицательной ориентации. Итак,

$$\lambda_1 = \frac{V(P, P_2, P_3, P_4)}{V(P_1, P_2, P_3, P_4)}, \quad \lambda_2 = \frac{V(P_1, P, P_3, P_4)}{V(P_1, P_2, P_3, P_4)}, \quad \lambda_3 = \frac{V(P_1, P_2, P, P_4)}{V(P_1, P_2, P_3, P_4)}, \quad \lambda_4 = \frac{V(P_1, P_2, P_3, P)}{V(P_1, P_2, P_3, P_4)}.$$

Аналогично предыдущим случаям  $\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 = 1$  и  $\lambda_i \geq 0, i = 1, 2, 3, 4$ . Для любой точки пространства также можно вычислить барицентрические координаты, но для точек вне тетраэдра будет нарушение условий  $\lambda_i \geq 0, i = 1, 2, 3, 4$ .

Все рассмотренные выше случаи позволяют заключить, что на основе барицентрических координат можно определять принадлежность точки симплексу.

#### 9.4. Пересечение луча с треугольником

При реалистичной визуализации основная операция – это пересечение луча с треугольником ненулевой площади.

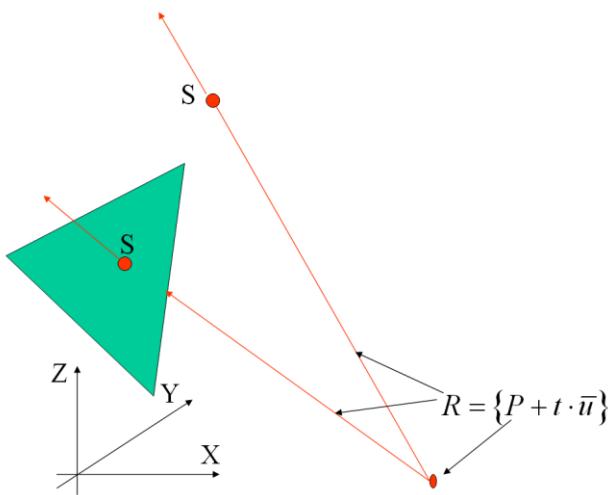


Рис. 9.8. Пересечение луча с треугольником

Итак, из точки  $P$  выпускаем луч в направлении  $\bar{u} : R(t) = \{P + t \cdot \bar{u}\}$ . И ищем его пересечение с треугольником  $\triangle P_1P_2P_3$  – точку  $S$ . Алгоритм состоит из следующих основных шагов.

1. Определяем плоскость треугольника. Для этого нам надо получить его нормаль, при помощи векторного произведения векторов сторон  $\overrightarrow{P_1P_2} \times \overrightarrow{P_1P_3}$ , нормируем и получаем  $|\bar{n}|=1$ . Если вектора  $\bar{u}$  и  $\bar{n}$  перпендикулярны –  $(\bar{u}, \bar{n})=0$  – луч параллелен плоскости треугольника, и пересечения нет, выход.
2. Находим точку  $S = R(t)$ , подставляя уравнение луча в уравнение плоскости:  $(R(t) - P_1, \bar{n}) = 0$ .
3. Треугольник  $\triangle P_1P_2P_3$  и точку  $S$  проецируем параллельно одной из координатных осей на одну из координатных плоскостей и получаем  $\triangle P'_1P'_2P'_3$  и  $S'$ . Для проецирования выбирается та координатная ось, чья координата в нормали  $\bar{n} = (n_x, n_y, n_z)$  максимальна по модулю,
4. Вычисляем барицентрические координаты  $S'$  относительно треугольника  $\triangle P'_1P'_2P'_3$ :
  - a. Вычисляем  $\lambda_1$ . Если это значение отрицательно, то пересечения нет, выход.
  - b. Вычисляем  $\lambda_2$ . Если это значение отрицательно, то пересечения нет, выход.
  - c. Если  $\lambda_1 + \lambda_2 > 1$ , то пересечения нет, выход.
  - d. Вычисляем  $\lambda_3 = 1 - (\lambda_1 + \lambda_2)$ .
5. Пересечение найдено:  $S = \lambda_1 P_1 + \lambda_2 P_2 + \lambda_3 P_3$ .

## 9.5. Разбиение единицы

Обратимся еще раз к выражению  $\lambda_1 P_1 + \dots + \lambda_n P_n$ , при  $\lambda_1 + \dots + \lambda_n = 1$  и  $\lambda_i \geq 0, i = 1, \dots, n$ . В каждом случае для симплексов мы имели минимальное множество опорных точек. А что будет, если мы на плоскости возьмем более трех, а в пространстве более четырех точек и попытаемся выяснить свойства точечного множества

$$\{ P = \lambda_1 P_1 + \dots + \lambda_n P_n, \lambda_1 + \dots + \lambda_n = 1 \text{ и } \lambda_i \geq 0, i = 1, \dots, n \}.$$

Числа  $\lambda_i$  представляют *разбиение единицы*. Докажем, что это множество – выпуклая оболочка, натянутая на опорные точки. Используем индукцию.

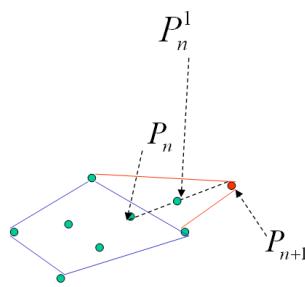


Рис. 9.9. О выпуклой оболочке и разбиении единицы

Для случая  $n=2$  это верно, см. выше про отрезок. Пусть утверждение верно для  $n$ , докажем, что оно верно для  $n+1$ . Итак, точка  $P = \sum_{i=1}^{n+1} \lambda_i \cdot P_i$  должна принадлежать выпуклой оболочке опорных точек. Сделаем ряд преобразований:

- $P = \sum_{i=1}^{n-1} \lambda_i \cdot P_i + \lambda_n \cdot P_n + \lambda_{n+1} \cdot P_{n+1}$ .
- $P = \sum_{i=1}^{n-1} \lambda_i \cdot P_i + (\lambda_n + \lambda_{n+1}) \cdot \left( \frac{\lambda_n}{\lambda_n + \lambda_{n+1}} P_n + \frac{\lambda_{n+1}}{\lambda_n + \lambda_{n+1}} \cdot P_{n+1} \right)$ .
- $P = \sum_{i=1}^{n-1} \lambda_i \cdot P_i + \lambda_n^1 \cdot P_n^1$ ,  $\sum_{i=1}^{n-1} \lambda_i + \lambda_n^1 = 1$  и  $0 \leq \lambda_n^1 \leq 1$ .

Что и требовалось доказать, см. также рис. 9.9.

Можно сделать вывод, что любая точка выпуклого многоугольника может быть получена при помощи такого построения.

## 9.6. Триангуляции

Триангуляцией называется представление плоской или трехмерной области совокупностью симплексов. При помощи триангуляций задаются поверхности в компьютерной графике. Треугольник – это самый популярный примитив при описании трехмерных сцен, поскольку в настоящее время графические акселераторы умеют обрабатывать только треугольники. Большое внимание исследованию и построению триангуляций с различными свойствами уделяется в таких областях, как вычислительная математика, географические информационные системы и др. Мы не будем подробно рассматривать эти вопросы, желающие могут познакомиться с ними в книге [23]. Отметим, что при построении триангуляции стремятся, чтобы все треугольники были как можно ближе к равномерным, а тетраэдры к правильным. Кстати с триангуляцией мы можем встретиться и в лесу, поскольку для представления рельефа поверхности земли устраиваются вполне телесные триангуляционные пункты.

Триангуляцией считается не произвольное разбиение области на симплексы, а обязательно должно выполняться правило: два симплекса триангуляции либо не пересекаются, либо пересекаются по симплексу меньшей размерности. На рис. 9.10 даны два примера неправильной и одной правильной стыковки треугольников.

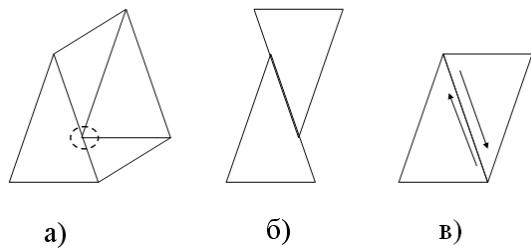


Рис. 9.9. а) вершина лежит на стороне, б) стороны частично пересекаются,  
в) правильнаястыковка

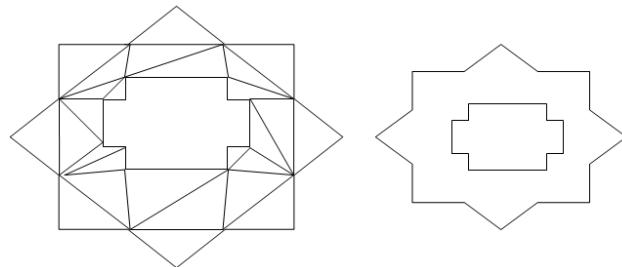


Рис. 9.11. Выделение границы объекта

Применение в триангуляции одинаково ориентированных симплексов позволяет более просто выполнять некоторые задачи. На рис. 9.10.в показаны направления обхода границы треугольников. Ясно, что если сторона является общей для двух треугольников, то она входит дважды – с различными ориентациями. На рис. 9.11 показан пример триангуляции многоугольника с дыркой. После того как были убраны все стороны, имеющие по 2 вхождения, результатом явилась граница многоугольника. Можно пойти дальше и дырки в области триангулировать симплексами другой ориентации, чем саму область.

## 10. Элементы геометрического моделирования

Данная проблема имеет большое значение во многих сферах человеческой деятельности: проектирование, дизайн, виртуальная реальность и т.д. В рамках данного курса нас будет интересовать задание форм кривых и поверхностей, например, чтобы сконструировать трехмерную сцену.

### 10.1. Конструирование кривых

Одна из классических задач вычислительной математики заключается в том, чтобы *аппроксимировать* кривую  $L$  другой кривой  $\tilde{L}$ , принадлежащей к заданному и хорошо изученному классу. Как правило, это требуется для сжатия информации об исходной кривой, передачи сжатой информации, и последующего ее восстановления (восстановления  $\tilde{L}$ ). В этом случае можно говорить о *конструировании* кривой, под которым понимать конструирование  $\tilde{L}$  при помощи задания сразу параметров "сжатой информации". Другими словами, конструктор формирует сжатую информацию о кривой, имея в голове представление о кривой  $L$ , а наблюдает восстановленную по этой информации кривую  $\tilde{L}$ . Очень часто сжатое представление включает последовательность точек исходной кривой  $L$ . Если на фазе восстановления требуется, чтобы  $\tilde{L}$  проходила через эти точки, то говорят о задаче *интерполяции*.

Рассмотрим ставшим уже классическим пример кубического многозвенника – кубического сплайна, проходящего через заданные точки [Завьялов]. Пусть  $L$  – это однозначная функция  $y = f(x)$ . Последовательность несовпадающих точек –  $\{y_i = f(x_i)\}_{i=1}^n$ . Требуется, чтобы:

- $\tilde{L}$  проходила через эти точки и на каждом отрезке  $[x_i, x_{i+1}], i = 1, \dots, n-1$  представляла кубический полином (звено);
- В узлах  $x_i, i = 2, \dots, n-1$  должна быть непрерывность первых и вторых производных.

Для однозначного определения  $\tilde{L}$  недостает 2 условия, в качестве которых можно задать, например, наклоны  $\tilde{L}$  на концах  $x_1$  и  $x_n$ , т.е. значения  $f'(x)$ .

Данный способ конструирования не позволяет *локальную модификацию*, поскольку при изменении любого из исходных данных (точки и наклоны) нам придется пересчитать всю кривую  $\tilde{L}$ . Это обозначает, что любое изменение данных о конструируемой прямой приводит к ее *глобальной модификации*. Естественно желание отлаживать вид кривой по частям, не возвращаясь без необходимости к уже построенным звеньям.

Даже на плоскости представление кривых в виде  $y = f(x)$  существенно сужает возможности конструктора, поэтому самым подходящим инструментом оказываются кривые, заданные в параметрической форме. Ранее мы уже сталкивались с параметрическим представлением отрезка и прямой.

### 10.2. Параметрические кривые

Сначала приведем некоторые сведения из дифференциальной геометрии. Кривая в пространстве  $R^3$  задается параметрически при помощи векторного выражения  $r(t) = \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix}$ ,  $t \in [a, b]$ . Этую векторную функцию можно дифференцировать по следующему правилу

$r'(t) = \begin{pmatrix} x'(t) \\ y'(t) \\ z'(t) \end{pmatrix}$ . Часто используется запись, принятая в кинематике  $\dot{r}(t) = \begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{z}(t) \end{pmatrix}$ . Очевидна

ориентация кривой: начало при  $t = a$ , конец при  $t = b$ . Аналогично можно говорить и о производных высших порядков.

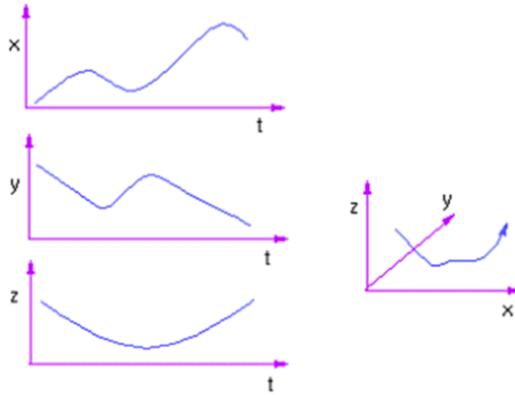


Рис. 10.1. Параметрическая кривая и ее проекции

На рис. 10.1. приведен пример параметрической кривой и графики ее проекций на координатные плоскости. Уже этот пример показывает, что конструирование по проекциям, используя визуальную обратную связь, является достаточно непростым делом. Как в теоретическом, так и в практическом плане, параметризации бывают хорошие и плохие. Наилучшей параметризацией является *натуральная* или *естественная*, когда параметр представляет собой длину дуги самой кривой, не вникаем в подробности. Приведем два случая параметризации окружности.

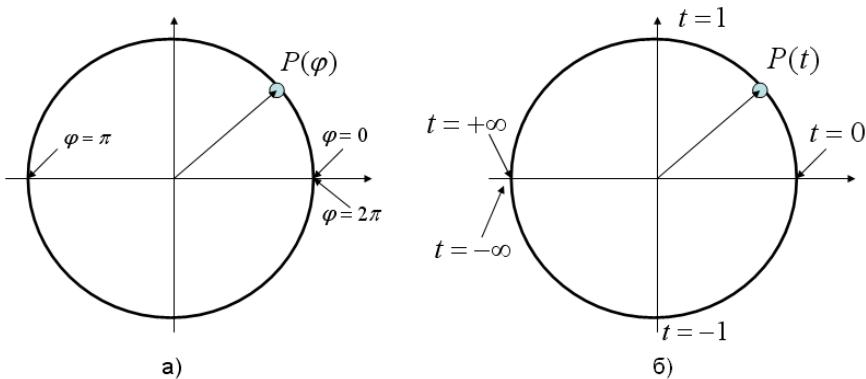


Рис. 10.2. Две параметризации окружности

На рис. 10.2.а изображена классическая параметризация окружности:

$$P(\varphi) = \begin{pmatrix} x(\varphi) \\ y(\varphi) \end{pmatrix} = \begin{pmatrix} \cos(\varphi) \\ \sin(\varphi) \end{pmatrix}, \quad 0 \leq \varphi < 2\pi. \quad (10.1)$$

А на рис. 10.2.б предлагается другая параметризация:

$$P(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} \frac{1-t^2}{1+t^2} \\ \frac{2at}{1+t^2} \end{pmatrix} \quad (10.2)$$

Предположим, что нам необходимо приблизить параметрическую кривую ломаной, у которой длина звена лежит в определенных пределах:  $k \leq \frac{l(t, t + \delta t)}{\delta t} \leq K$ . Это необходимо как

при растеризации кривых, так и при подготовке программ для станков с числовым управлением. Вспомним, что длина участка кривой выражается формулой:

$$l(a, b) = \int_a^b |r'(t)| \cdot dt. \quad (10.3)$$

Для параметризации (10.1) имеем  $\frac{l(t, t + \delta t)}{\delta t} = \pi$ , т.е. теперь мы можем легко выбрать шаг по

параметру и получить вершины аппроксимирующей ломаной. А вот для второго случая, представленного формулой (10.2), мы не найдем нижней границы  $k$  для любого значения  $\delta t$ .

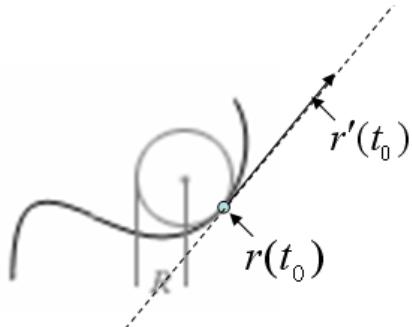


Рис. 10.3. Радиус кривизны и касательная

Очевидно, что на параметризацию влияет то, в какой системе координат мы определяем векторное уравнение кривой. У пространственных кривых имеются инварианты – это графики *кривизны* и *кручения*. Для кривой  $r(t)$  кривизной в точке  $t = t_0$  называется величина

$k_1$ , которая вычисляется согласно следующему соотношению:  $k_1^2 = \frac{1}{R^2} = \frac{|\dot{r} \times \ddot{r}|^2}{|\dot{r}|^3}$  или

$$k_1^2 = \frac{\left| \begin{matrix} \ddot{x} & \ddot{y} \\ \dot{x} & \dot{y} \end{matrix} \right|^2 + \left| \begin{matrix} \ddot{y} & \ddot{z} \\ \dot{y} & \dot{z} \end{matrix} \right|^2 + \left| \begin{matrix} \ddot{z} & \ddot{x} \\ \dot{z} & \dot{x} \end{matrix} \right|^2}{(\dot{x}^2 + \dot{y}^2 + \dot{z}^2)^3}. \text{ Величина } k_1 \text{ обратна радиусу кривизны } R, \text{ см. рис. 10.3.}$$

Кручение  $k_2$  определяется согласно соотношениям:  $k_2(t) = -\frac{(\dot{r}, \ddot{r}, \ddot{r})}{|\dot{r} \times \ddot{r}|^2}$ , в числителе смешанное

произведение  $(\dot{r}, \ddot{r}, \ddot{r}) = (\dot{r}, \ddot{r} \times \ddot{r})$ . Простой анализ этих двух величин показывает, что прямая, имеющая нулевую кривизну, должна удовлетворять условию  $\ddot{r}(t) \equiv 0$ . Кривая является плоской, если у нее нулевое кручение:  $k_2(t) \equiv 0$ .

И, наконец, введем необходимую нам касательную к параметрически заданной кривой  $r(t)$  в точке  $r(t_0)$  – это прямая  $T(u) = r(t_0) + u \cdot \dot{r}(t_0)$  – пунктир на рис. 10.3,  $u$  – параметр.

### 10.3. Построение звена кривой

Кривую с локальной модификацией будем строить, начиная с отдельного звена, будем строить в параметрическом виде. Между этими точками каждая координатная компонента  $x(t), y(t), z(t)$  должна быть полиномом степени не ниже 3-х. Почему 3-я степень? Потому что мы строим *пространственную кривую*. Значит, она имеет ненулевое кручение, и производная  $\ddot{r}(t)$  не должна быть нулевой, а степень 3 – это минимальная степень, удовлетворяющая этим условиям. Традиционно звено или сегмент кривой описывается как  $r(t), t \in [0, 1]$ . Давайте распишем подробнее:

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x,$$

$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y,$$

$$z(t) = a_z t^3 + b_z t^2 + c_z t + d_z.$$

У нас 12 неизвестных. 2 концевые точки дают 6 условий. Каковы дополнительные условия? Рассмотрим два наиболее известные подхода. В дальнейшем мы будем рассматривать только  $x(t)$ , поскольку все выкладки для остальных двух координат проводятся аналогично и независимо.

### 10.3.1. Сегмент кривой в форме Эрмита

Предлагается недостающие 6 условий задавать при помощи касательных векторов  $R_1$  и  $R_4$  в концевых точках  $P_1$  и  $P_4$ , см. рис. 10.4.



Рис. 10.4. Сегмент в форме Эрмита

Итак,

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x, \quad (10.4)$$

тогда

$$\frac{dx}{dt} = 3a_x t^2 + 2b_x t + c_x. \quad (10.5)$$

С учетом интервала изменения параметра  $[0,1]$  мы можем записать начальные условия:

$$x(0) = P_{1x}, x'(0) = R_{1x}, x(1) = P_{4x}, x'(1) = R_{4x}. \quad (10.6)$$

Замечание. При помощи нижнего индекса  $(.)_x$  будем обозначать  $x$ -овую компоненту того или иного объекта.

Перепишем (10.4) в матричном виде и получим

$$x(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}_x = T C_x. \quad (10.7)$$

А (10.5) будет выглядеть следующим образом:

$$x'(t) = \begin{bmatrix} 3t^2 & 2t & t & 0 \end{bmatrix} C_x. \quad (10.8)$$

Аналогично переписываем начальные условия (10.6):

$$P_{1x} = x(0) = [0 \ 0 \ 0 \ 1] C_x,$$

$$P_{4x} = x(1) = [1 \ 1 \ 1 \ 1] C_x,$$

$$R_{1x} = x'(0) = [0 \ 0 \ 1 \ 0] C_x,$$

$$R_{4x} = x'(1) = [3 \ 2 \ 1 \ 0] C_x.$$

Таким образом, переходим к матричной записи уравнения:

$$\begin{pmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{pmatrix}_x = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} C_x.$$

Решаем и получаем:

$$C_x = \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{pmatrix}_x = M_h G_{hx},$$

Где  $M_h$  называется Эрмитовой матрицей, а  $G_h$  – Эрмитовым геометрическим вектором. Подставляя в (10.7) окончательно получаем:

$$\begin{aligned} x(t) &= TM_h G_{hx}, \\ y(t) &= TM_h G_{hy}, \\ z(t) &= TM_h G_{hz}. \end{aligned} \quad (10.9)$$

Или в более короткой форме:  $r(t) = TM_h G_h$ .

### 10.3.2. Сегмент кривой в форме Безье

Эта форма является простой переформулировкой формы Эрмита. Недостающие 6 условий задаются при помощи специальных контрольных или управляемых точек  $R_2$  и  $R_3$ , которые берутся на касательных в концевых точках  $P_1$  и  $P_4$ , см. рис. 10.5.

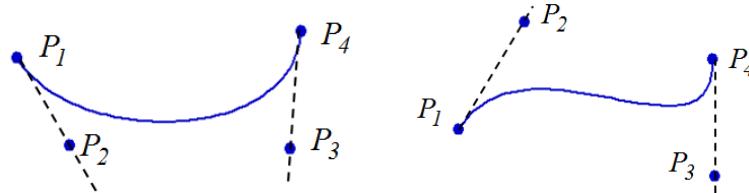


Рис. 10.5. Сегменты кривой в форме Безье

В отличие от концевых точек управляемые точки не обязательно лежат на кривой и выбираются исходя из следующих условий:

$$R_1 = r'(0) = 3(P_2 - P_1),$$

$$R_4 = r'(1) = 3(P_4 - P_3).$$

На основе этих условий мы можем определить следующее равенство:

$$G_h = \begin{pmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{pmatrix} \begin{pmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{pmatrix} = M_{hb} G_b,$$

которое представляет Эрмитов геометрический вектор через матрицу перехода  $M_{hb}$  и геометрический вектор Безье  $G_b$ . Подставляем данное выражение в (10.9) и получаем окончательно:

$$x(t) = T \cdot (M_h \cdot M_{hb}) \cdot G_{bx} = T \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} G_{bx} = TM_b G_{bx}. \quad (10.10)$$

Раскрывая правую часть и приводя подобные, мы можем записать уравнение сегмента кривой в форме Безье в следующей форме:

$$r(t) = (1-t)^3 P_1 + 3(1-t)^2 t P_2 + 3(1-t)t^2 P_3 + t^3 P_4, \quad (10.11)$$

которая более удобная и наглядная для проведения вычислений. Кроме того на основе этой записи нам можно сделать очень важный вывод по поводу геометрии полученного сегмента. Действительно, все коэффициенты при точках неотрицательные при  $0 \leq t \leq 1$ , а их сумма равна 1, т.к.

$$(1-t)^3 + 3(1-t)^2 t + 3(1-t)t^2 + t^3 = B_1(t) + B_2(t) + B_3(t) + B_4(t) = [(1-t)+t]^3 = 1. \quad (10.12)$$

В связи с этим используем факт, что это разбиение единицы, а, следовательно, все точки сегмента кривой лежат внутри выпуклой оболочки, натянутой на исходные 4 точки, т.е. внутри *тетраэдра*, см. рис. 10.6.

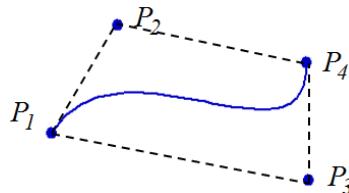


Рис. 10.6. Выпуклая оболочка сегмента кривой Безье

Сопрягающие функции  $B_i(t)$  – это полиномы Бернштейна. Чем больше значение одной из этих функций, тем больше влияние соответствующей точки на результат, т.е. тем ближе  $r(t)$  к этой контрольной точке.

Сегменты Безье с их контрольными точками были найдены более удобными по сравнению с касательными формами Эрмита для проектирования сегмента в интерактивном режиме с визуальной обратной связью. Тем не менее, форма Эрмита может применяться в тех случаях, когда начальный вид кривой известен и описан аналитически. Тогда на первом этапе используется форма Эрмита, а затем для дальнейшего редактирования – форма Безье.

Отметим, что существуют кривые Безье более высоких порядков.

*Упражнение.* Показать, что любой кусок сегмента Безье можно перепараметризовать так, чтобы параметр изменялся от 0 до 1.

### 10.3.3. Самопересечения сегмента кривой в форме Безье

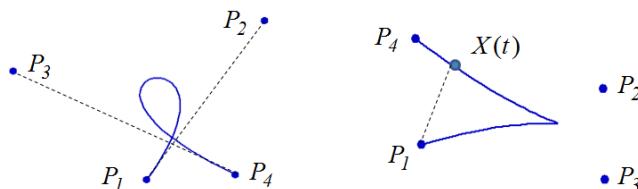


Рис. 10.7. Самопересечение сегмента Безье

На рис. 10.7. показан случай самопересечения сегмента кривой Безье, который вполне понятен, исходя из свойства притяжения контрольными точками. Тем не менее, артефакты в виде петель и заострений крайне нежелательны на практике. Если мы можем предложить формальные правила, обеспечивающие сегменты без петель, то эти правила можно учесть в редакторе, который не позволит пользователю их нарушать.

*Одно из правил.* Пусть числа  $\alpha, \beta$  таковы, что  $P_1P_2 = \alpha P_1X(t)$ ,  $P_3P_4 = \alpha X(t)P_4$ . Тогда кривая сама себя пересекает при  $0 < t < 1$ , если существует точка  $X(t)$ , такая что  $(\alpha - \frac{4}{3})(\beta - \frac{4}{3}) > \frac{4}{9}$ , см. рис. 10.7 [Fox Pratt].

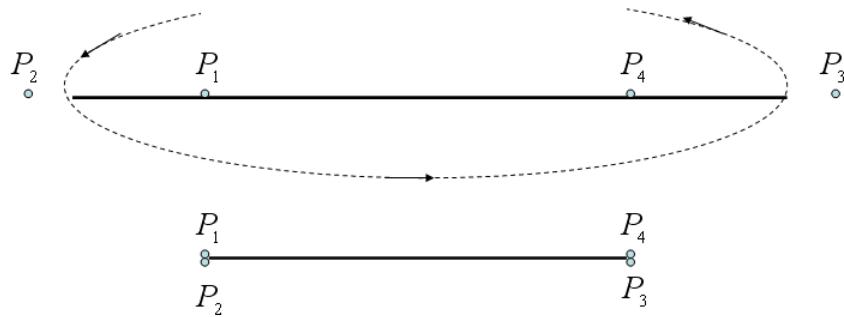


Рис. 10.8. Петли (вверху), без петель (внизу)

Форрест (1968) предложил пользоваться более просто проверяемым условием: хорды  $P_1P_2$  и  $P_3P_4$  не должны превышать хорды  $P_1P_4$ . Кстати, точные равенства достигаются на отрезке – см. рис. 10.8. Здесь верхний отрезок имеет достаточно сложную структуру: а) сначала точка  $r(t)$  пробегает его от точки  $P_1$  в сторону точки  $P_2$ , б) затем ослабевает влияние управляемой точки  $P_2$  и точка  $r(t)$  снова проходит начальный участок до  $P_1$ ; в) двигается дальше.

У нижнего отрезка на рис. 10.8 нет дублирования точек, поскольку исходно заданы:  $P_1 = P_2$  и  $P_3 = P_4$ .

*Упражнение.* Записать параметрическое представление для отрезков, представленных на рис. 10.8.

#### 10.3.4. Геометрическое построение сегмента кривой в форме Безье

Форма Безье действительно очень удобна для рисования в диалоговом режиме. В большинстве редакторов пользователь может построить отрезок прямой, а может те же две точки соединить сегментом плоской кривой Безье. Для плоской кривой (всего три точки: две концевые и одна контрольная) достаточно одной дополнительной управляемой точки. Рассмотрим серию этапов вычисления точки согласно рис. 10.9.

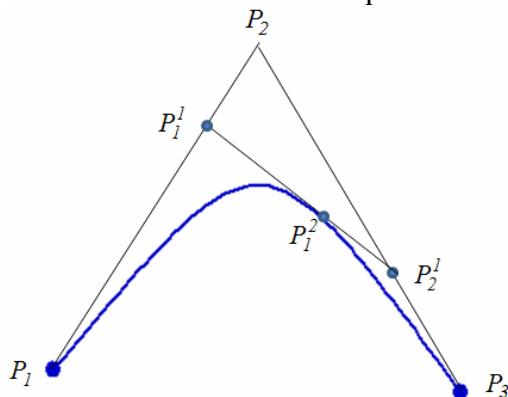


Рис. 10.9. Вычисление точки  $P(t)$

1. Исходные данные: концевые точки  $P_1$  и  $P_3$ , управляемая точка  $P_2$ .
2. Вычисляем  $P_1^1 = (1-t)P_1 + tP_2$ .
3. Вычисляем  $P_2^1 = (1-t)P_2 + tP_3$ .
4. Вычисляем  $P(t) = P^2 = (1-t)P_1^1 + tP_2^1$ .
5. Действительно:  $P(t) = P^2 = (1-t)^2 P_1 + (1-t)tP_2 + t^2 P_3$ .

И аналогичные построения для пространственного случая показаны на рис. 10.10.

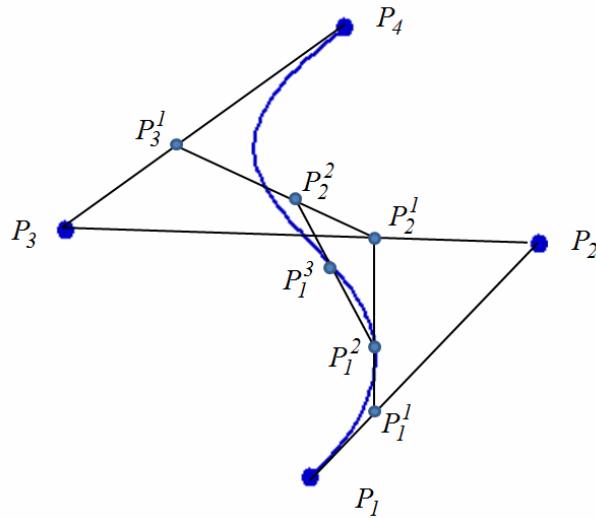


Рис. 10.10. Вычисление точки  $P(t)$

1. Исходные данные: концевые точки  $P_1$  и  $P_4$ , управляющие точки  $P_2$  и  $P_3$ .
2. Вычисляем  $P_1^1 = (1-t)P_1 + tP_2$ .
3. Вычисляем  $P_2^1 = (1-t)P_2 + tP_3$ .
4. Вычисляем  $P_3^1 = (1-t)P_3 + tP_4$ .
5. Вычисляем  $P_1^2 = (1-t)P_1^1 + tP_2^1$ .
6. Вычисляем  $P_2^2 = (1-t)P_2^1 + tP_3^1$ .
7. Вычисляем  $P_1^3 = (1-t)P_1^2 + tP_2^2$ .

Подставим и получим  $P(t) = P_1^3 = (1-t)^3 P_1 + 3(1-t)^2 t P_2 + 3(1-t)t^2 P_3 + t^3 P_4$ .

### 10.3.5. Некоторые свойства кривых Безье

Сегменты кривой в форме Безье могут иметь и более четырех опорных или контрольных точек, так кривая, построенная на  $n+1$  опорной точке (включая концы), определяется следующим образом:

$$P^n(t) = \sum_{i=0}^n P_i \cdot B_i^n(t),$$

где  $B_i^n(t) = C_i^n \cdot t^i (1-t)^{n-i} = \frac{n!}{i!(n-i)!} \cdot t^i (1-t)^{n-i}$  – полиномы Берштейна.

Отмечаются следующие свойства:

- степень многочлена, представляющего кривую в аналитическом виде на 1 меньше числа опорных точек;
- кривая Безье проходит через первую и последнюю управляющие точки;
- инвариантность относительно аффинных преобразований;
- инвариантность относительно линейных замен параметризации;
- кривая Безье принадлежит выпуклой оболочке опорных точек, что следует из геометрического способа построения;

- симметричность: если рассматривать управляющие точки в противоположном порядке, то кривая не изменится.

## 10.4. Стыковка звеньев кривой

Оба метода интерполяции предполагают, что целую кривую надо строить из небольших сегментов. При интерактивной работе можно остановиться на форме кривой удовлетворяющей визуально. Правда, кривые конструируются не только для дизайна, но и для расчетов. Требуется удовлетворить некоторые условия сочленения кусков, как правило, это требование непрерывности кривой и касательной.

Итак, если мы строим *непрерывную* кривую  $L$ , состоящую из  $n$  сегментов  $L_i, i=1,\dots,n$ , то параметр  $t$  будет пробегать значения от 0 до  $n$ . Тогда вычисление точки кривой  $L(t)$  при  $t \in [0, n]$  сводится к следующему набору действий:

- Определяем сегмент кривой  $L_i$ , такой что  $i-1 \leq t < i$ .
- Вычисляем  $L(t) = L_i(t - i + 1)$ .

В случае, когда  $L$  разрывная в узлах, т.е.стыковка сегментов не непрерывна, данный алгоритм также работает.

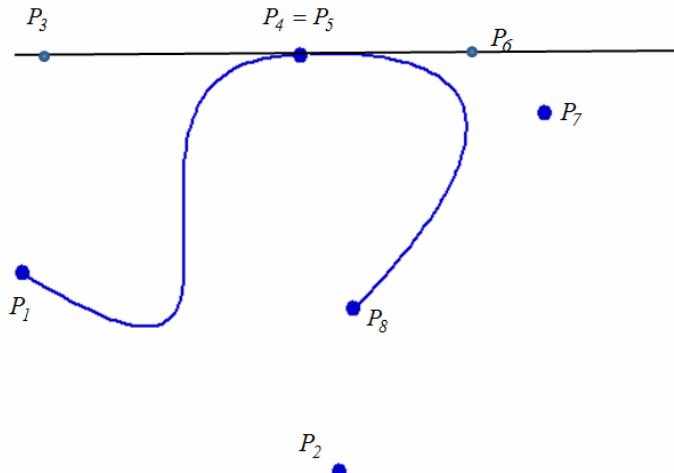


Рис. 10.11. Гладкая стыковка сегментов Безье

Пусть сегмент Эрмита  $L_i$  задается точками  $P_1, P_4$  и касательными  $R_1, R_4$ , а сегмент  $L_{i+1}$  – при помощи  $P'_1, P'_4, R'_1, R'_4$ .

- Если  $P_4 = P'_1$ , то стыковка сегментов обеспечивает непрерывность.
- Если, кроме того,  $R_4 = k \cdot R'_1, k > 0$ , то обеспечивается непрерывность касательной.

Аналогично и для сегментов кривых Безье. Пусть сегмент Безье  $L_i$  задается точками  $P_1, P_2, P_3, P_4$ , а сегмент  $L_{i+1}$  – при помощи  $P_5, P_6, P_7, P_8$ , см. рис. 10.11.

- Для непрерывности кривой Безье необходимо, чтобы  $P_4 = P_5$ .
- Для непрерывности касательной нужно, чтобы точки  $P_3, P_4$  и  $P_5, P_6$  лежали на прямой, т.е. выполнялось условие  $P_6 - P_5 = k(P_4 - P_3), k > 0$ .

*Упражнение.* Нарисуйте пример для гладкой стыковки кусков в форме Эрмита аналогично рис. 10.10.

## 10.5. В-сплайны

В случае сегментов Эрмита мы обеспечивали прохождение кривой через обе заданные концевые точки, управляя формой сегмента с помощью касательных в концевых точках. В случае сегмента Безье кривая проходит через концевые точки и не проходит через другие две точки, управляющие формой сегмента.

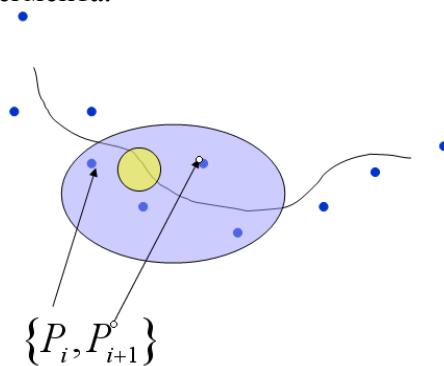


Рис. 10.12. Сегмент В-сплайна

Мы рассмотрим В-сплайн несколько под другим углом, чем это делается в теории интерполяции, а именно: с точки зрения конструирования локально модифицируемой кривой. В отличие от форм Эрмита и Безье В-сплайн – это еще более трудно видимая зависимость от исходных данных (точек), точнее, кривая не проходит через исходные точки, как правило, но они используются как управляющие для создания ее формы. Эта кривая строится на основе сразу  $n$  точек  $P_1, \dots, P_n$ , см. рис. 10.12. Перейдем сразу к расчетным формулам.

Кривая имеет несколько сегментов, каждый из которых  $i$ -й будем определять *основной* опорной точкой  $P_i$ ,  $i = 2, \dots, n-2$ . Для вычисления точек  $i$ -го сегмента используется информация о точках  $P_{i-1}, P_i, P_{i+1}, P_{i+2}$ , они составляют геометрический вектор  $G_s^i$   $i$ -го сегмента В-сплайна. Точки сегмента  $L_i$  вычисляются по формуле (в обозначениях п. 10.3.1) :

$$x_i(t) = T \cdot M_s \cdot G_s^i, \text{ где } M_s = \frac{1}{6} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{pmatrix}, G_s^i = \begin{pmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{pmatrix}, t \in [0,1]. \quad (10.13)$$

Отметим следующие свойства В-сплайновой кривой:

- Она непрерывна.
- Непрерывна первая производная в точках стыковки участков, т.е. непрерывна касательная.
- Непрерывна вторая производная в точках стыковки участков, т.е. непрерывна и кривизна.
- $i$ -й участок кривой целиком лежит внутри выпуклой оболочки 4-х точек  $P_{i-1}, P_i, P_{i+1}, P_{i+2}$  аналогично участку кривой Безье.
- Парная точка уменьшает степень гладкости стыковки участков на 1, тройная точка уменьшает на 2 порядка.

## 10.6. Геометрическая и параметрическая непрерывность

Для параметрически заданных кривых различаются геометрическая и параметрическая непрерывность. Определены следующие типы геометрической непрерывности:

- $G0$  – кривая непрерывная, т.е. куски стыкуются.
- $G1$  – касательные имеют одинаковое направление, но не величину, т.е. первые производные пропорциональны.
- $G2$  – первые и вторые производные пропорциональны в точке стыковки кусков.

Параметрическая непрерывность – это гладкость кривой в обычном смысле, т.е.  $Cn$  означает равенство производных (учесть векторную запись) до  $n$ -ой включительно,  $n = 0, 1, \dots$

## 10.7. Конструирование поверхностей

При конструировании поверхностей мы также обратим внимание на возможность построения поверхностей и локально их модифицировать, т.е. строить поверхности из *кусков* (patches). Действительно, после того как дизайнер кузова закончил работу с капотом и перешел к отладке формы крыши или багажника, он должен быть уверен, что его текущие действия не затронут далекие области корпуса. Аналогично кривым мы начнем с того, что научимся создавать отдельные куски поверхности. Один из первых подходов к решению этой задачи связан с именем Кунса.

### 10.7.1. Параметрические поверхности

В отличие от кривых, которые описываются с помощью одного параметра  $t$ , поверхности описываются при помощи двух независимых параметров  $u$  и  $v$ . Любая точка

параметрической поверхности определяется как  $r(u, v) = \begin{pmatrix} x(u, v) \\ y(u, v) \\ z(u, v) \end{pmatrix}, (u, v) \in D \subset \mathbb{R}^2$ . И, исходя

из подобных же соображений как для кривых, требуется, чтобы отдельный кусок задавался на квадрате  $(u, v) \in [0, 1] \times [0, 1]$ . Особое значение имеют *параметрические линии*, линии, на которых  $u$  или  $v$  постоянны, см. рис. 10.13.

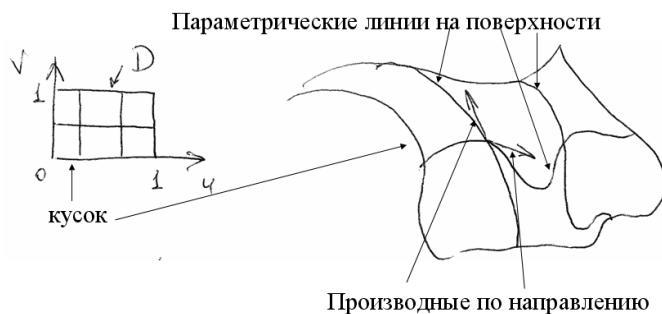


Рис. 10.13. Параметрические поверхности

$$\text{Производные по направлениям } r_u(u, v) = \begin{pmatrix} \frac{\partial x(u, v)}{\partial u} \\ \frac{\partial y(u, v)}{\partial u} \\ \frac{\partial z(u, v)}{\partial u} \end{pmatrix} \quad \text{и} \quad r_v(u, v) = \begin{pmatrix} \frac{\partial x(u, v)}{\partial v} \\ \frac{\partial y(u, v)}{\partial v} \\ \frac{\partial z(u, v)}{\partial v} \end{pmatrix} \quad \text{определяют}$$

касательные вдоль параметрических линий. Нормаль к поверхности задается формулой:  $n(u, v) = r_u(u, v) \times r_v(u, v)$ . Рис. 10.14 показывает, что любой кривой на прямоугольнике,

представляющем область изменения параметров, соответствует кривая на параметрической поверхности.

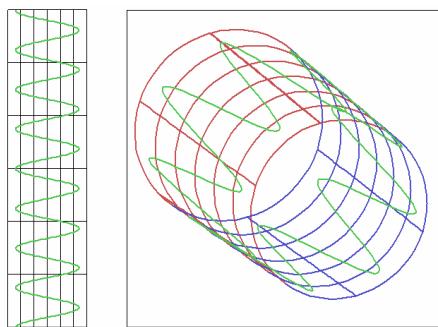


Рис. 10.14. Отображение кривой из области параметров на поверхность

Данных сведений нам будет достаточно для дальнейшего изложения материала, более подробно см. [Погорелов, Фокс].

### 10.7.2. Поверхности Кунса

Изложим последовательность построения куска поверхности по методу Кунса.

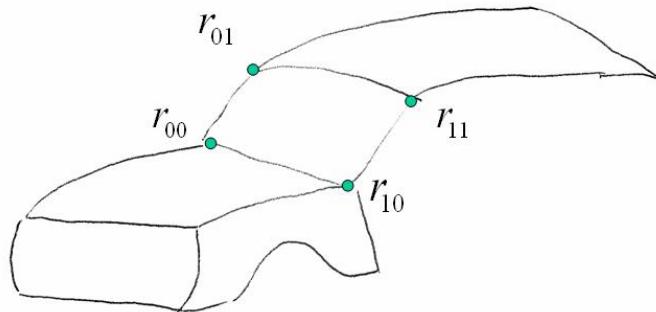


Рис. 10.15. Конструирование куска

Пусть конструктор имеет в голове начальный вид кузова, как тот представлен на рис. 10.15. Кузов уже мысленно разбит на куски. И пусть, он начинает с лобового стекла. Первое, что необходимо задать, это 4 точки "углов" куска:  $r_{00}, r_{01}, r_{10}, r_{11}$ . Первое, что часто делается, – это билинейная интерполяция, т.е. по этим точкам строится следующий кусок поверхности:

$$BL(u, v) = (1-u)(1-v)r_{00} + u(1-v)r_{10} + (1-u)v r_{01} + u v r_{11}, \quad (u, v) \in [0, 1]^2.$$

Эта поверхность известна как седло и, очевидно, что устроит она в редких случаях. Следовательно, нужна дополнительная информация. Пусть это будет форма граничных кривых:

$r_1(u)$  – от  $r_{00}$  до  $r_{10}$ ;

$r_2(u)$  – от  $r_{01}$  до  $r_{11}$ ;

$s_1(v)$  – от  $r_{00}$  до  $r_{01}$ ;

$s_2(v)$  – от  $r_{10}$  до  $r_{11}$ .

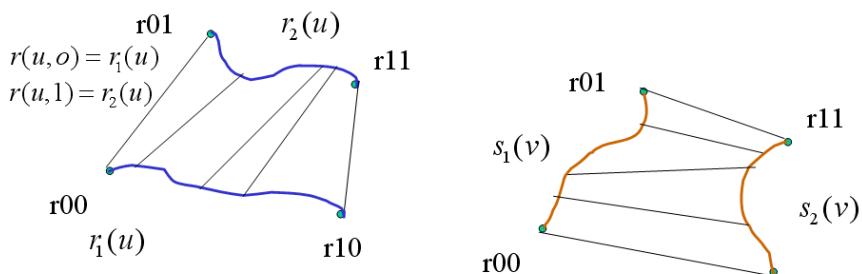


Рис. 10.16. Линейчатые поверхности на основе граничных кривых

На следующем этапе строим две линейчатые поверхности, см. рис. 10.16:

$$Lv(u, v) = (1-v) \cdot r_1(u) + v \cdot r_2(u) \text{ и } Lu(u, v) = (1-u) \cdot s_1(v) + u \cdot s_2(v).$$

Если теперь сложить эти две поверхности, то оказывается, что на границах и в углах у нас не получаются исходные данные: граничные кривые и угловые точки. Оказывается, что "излишки" – это не что иное, как билинейная форма, построенная выше. И, окончательно, кусок Кунса определяется по формуле:

$$r(u, v) = Lu(u, v) + Lv(u, v) - BL(u, v).$$

### 10.7.3. Задание куска в форме Эрмита

Итак, мы строим кусок поверхности, пока отдельно от других,  $r(u, v)$ ,  $(u, v) \in [0, 1]^2$ . Опять выбираем кубический по каждому параметру (бикубический) полином для представления отдельной координаты:  $x(u, v) = \sum_{i,j=1}^4 a_{i,j} u^{4-i} v^{4-j}$ . Запишем короче:  $x(u, v) = U \cdot C_x \cdot V^T$ , где:

$U = [u^3 \ u^2 \ u \ 1]$ ,  $V = [v^3 \ v^2 \ v \ 1]$ . Как и в одномерном случае записи для двух других координат выглядят аналогично:  $y(u, v) = U \cdot C_y \cdot V^T$ ,  $z(u, v) = U \cdot C_z \cdot V^T$ . Надо использовать 16 условий для определения 16 коэффициентов. В одномерном случае для участка кривой в форме Эрмита требуется 2 точки и 2 вектора касательных. Вспоминаем, что  $C_x$  – вектор

$$\text{коэффициентов при координате } x: C_x = \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{pmatrix}_x = M_h G_{hx}, \quad x(u) = UM_h G_{hx}.$$

Тогда

$$x(u, v) = UM_h G_{hx}(v) = UM_h \begin{pmatrix} P_1(v) \\ P_4(v) \\ R_1(v) \\ R_4(v) \end{pmatrix}_x. \quad (10.14)$$

Здесь:  $P_1(v)$  – это кривая на границе  $u = 0$ ,  $P_4(v)$  – кривая на границе  $u = 1$ . Соответственно  $R_1(v)$  и  $R_4(v)$  – это вектора касательных (по направлению  $v$ ) к поверхности в точках этих кривых.

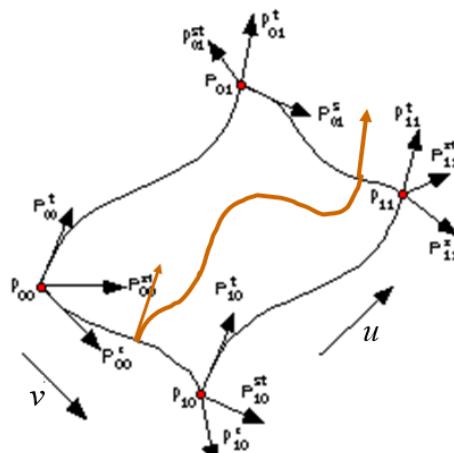


Рис. 10.17. Кривая на поверхности с касательными  $R_1(v)$  и  $R_4(v)$

Ну а теперь запишем все эти  $P_1(v)$ ,  $P_4(v)$ ,  $R_1(v)$ ,  $R_4(v)$ , тоже применяя форму Эрмита.

$$P_{1x}(v) = V \cdot M_h \cdot \begin{pmatrix} q_{11} \\ q_{12} \\ q_{13} \\ q_{14} \end{pmatrix}_x, \quad P_{4x}(v) = V \cdot M_h \cdot \begin{pmatrix} q_{21} \\ q_{22} \\ q_{23} \\ q_{24} \end{pmatrix}_x,$$

$$R_{1x}(v) = V \cdot M_h \cdot \begin{pmatrix} q_{31} \\ q_{32} \\ q_{33} \\ q_{34} \end{pmatrix}_x, \quad R_{4x}(v) = V \cdot M_h \cdot \begin{pmatrix} q_{41} \\ q_{42} \\ q_{43} \\ q_{44} \end{pmatrix}_x.$$

Таким образом, 4 кубических полинома представляем как

$$[P_1(v) \ P_4(v) \ R_1(v) \ R_4(v)]_x = VM_h \begin{pmatrix} q_{11} & q_{21} & q_{31} & q_{41} \\ q_{12} & q_{22} & q_{32} & q_{42} \\ q_{13} & q_{23} & q_{33} & q_{43} \\ q_{14} & q_{24} & q_{34} & q_{44} \end{pmatrix}_x.$$

Транспонируем и получаем:

$$\begin{bmatrix} P_1(v) \\ P_4(v) \\ R_1(v) \\ R_4(v) \end{bmatrix}_x = \begin{pmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{21} & q_{22} & q_{23} & q_{24} \\ q_{31} & q_{32} & q_{33} & q_{34} \\ q_{41} & q_{42} & q_{43} & q_{44} \end{pmatrix} M_h^T V^T = Q_x M_h^T V^T.$$

Подставляем полученное выражение в (10.14) и получаем:

$$\begin{aligned} x(u, v) &= U \cdot M_h \cdot Q_x \cdot M_h^T \cdot V^T, \\ y(u, v) &= U \cdot M_h \cdot Q_y \cdot M_h^T \cdot V^T, \\ z(u, v) &= U \cdot M_h \cdot Q_z \cdot M_h^T \cdot V^T. \end{aligned} \quad (10.15)$$

Как определить элементы матриц  $Q_x$ ,  $Q_y$ ,  $Q_z$ , с помощью точек и векторов касательных?

Достаточно формально:

- $q_{11} = x(0, 0) = x_{00}$ , т.к. это начало  $P_{1x}(v) = r(0, v)$ .
- $q_{12} = x(0, 1) = x_{01}$ , т.к. это конец  $P_{1x}(v) = r(1, v)$ .
- $q_{13} = \frac{\partial x}{\partial v}(0, 0) = x_{v,00}$ , т.е. наклон в начале для кривой  $P_{1x}(v) = r(0, v)$ .
- $q_{14} = \frac{\partial x}{\partial v}(0, 1) = x_{v,01}$ , т.е. наклон в начале для кривой  $P_{1x}(v) = r(0, v)$ .

Этого вполне достаточно для определения граничной кривой  $P_{1x}(v) = r(0, v)$ . Понадобятся и вторые производные типа:  $q_{33} = \frac{\partial^2 x}{\partial u \cdot \partial v}(0, 0) = x_{uv,00}$ ; которая задает начальный наклон для графика производной – касательной  $R_{1x}(v)$ . Используя введенные мнемонические обозначения, запишем всю матрицу:

$$Q_x = \begin{pmatrix} x_{00} & x_{01} & x_{v,00} & x_{v,01} \\ x_{10} & x_{11} & x_{v,10} & x_{v,11} \\ x_{u,00} & x_{u,01} & x_{uv,00} & x_{uv,01} \\ x_{u,10} & x_{u,11} & x_{uv,10} & x_{uv,11} \end{pmatrix}.$$

С этой формой связаны также имена Кунса и Фергюссона.

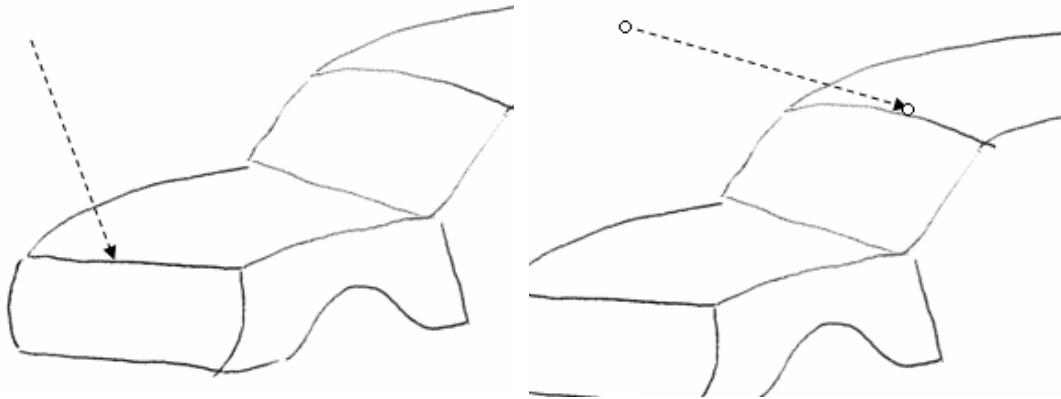


Рис. 10.18. Сшивка кусков: непрерывность (слева), гладкость (справа)

Теперь, чтобы сшить вместе два куска по общей граничной кривой  $r_2(u)$  первого куска, достаточно соблюсти следующее условие (см. рис. 10.18):

$$Q^1 = \begin{pmatrix} * & * & * & * \\ q_{21} & q_{22} & q_{23} & q_{24} \\ * & * & * & * \\ * & * & * & * \end{pmatrix} \Leftrightarrow \begin{pmatrix} q_{21} & q_{21} & q_{21} & q_{21} \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix} = Q^2.$$

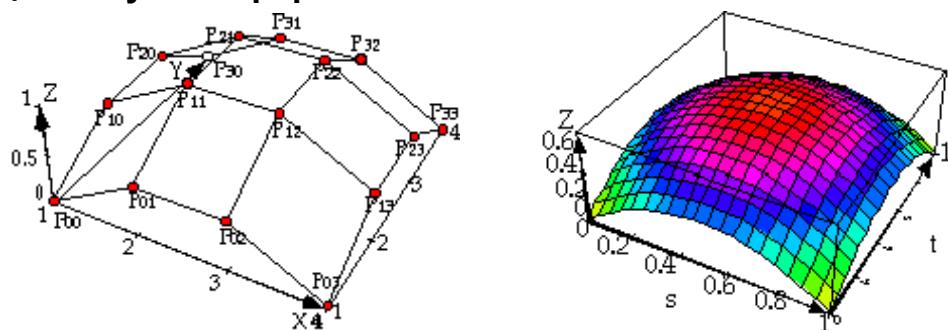
Для того, чтобы при этом обеспечить гладкость на граничной кривой при переходе от куска к куску, необходимо выполнить еще одно условие:

$$Q^1 = \begin{pmatrix} * & * & * & * \\ q_{21} & q_{22} & q_{23} & q_{24} \\ * & * & * & * \\ q_{41} & q_{42} & q_{43} & q_{44} \end{pmatrix} \Leftrightarrow \begin{pmatrix} q_{21} & q_{21} & q_{21} & q_{21} \\ * & * & * & * \\ k \cdot q_{41} & k \cdot q_{42} & k \cdot q_{43} & k \cdot q_{44} \\ * & * & * & * \end{pmatrix} = Q^2, \text{ где } k > 0.$$

Если сшивка производится по другой стороне, т.е. по кривой  $s_2(v)$  первого куска, то вместо условий на строки мы должны выполнить те же условия на столбцы. Чем хорош данный способ задания кусков поверхности? Тем, что в качестве исходных форм кусков можно взять какую-либо поверхность с известным аналитическим заданием. Затем, меняя параметры матриц, подгонять куски к требуемому виду.

Чем он плох? Трудно мыслить в касательных и особенно в смешанных производных. Поэтому, как и в случае кривых, обратимся к форме Безье.

#### 10.7.4. Задание куска в форме Безье



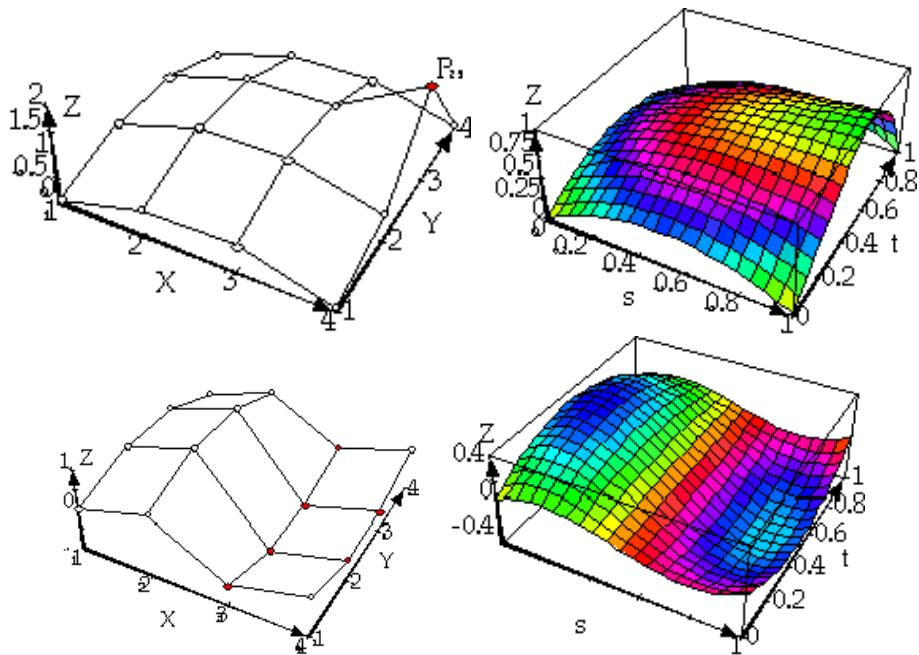


Рис. 10.19. Управляющие точки куска Безье и кусок поверхности

Обобщая формулы для кривых Безье, можем записать, что любая точка куска параметрической поверхности в форме Безье может быть представлена выражением:

$$P(u, v) = U \cdot M_b \cdot P \cdot M_b^T \cdot V^T = \begin{pmatrix} x(u, v) \\ y(u, v) \\ z(u, v) \end{pmatrix} = \begin{pmatrix} U \cdot M_b \cdot P_x \cdot M_b^T \cdot V^T \\ U \cdot M_b \cdot P_y \cdot M_b^T \cdot V^T \\ U \cdot M_b \cdot P_z \cdot M_b^T \cdot V^T \end{pmatrix}, (u, v) \in [0, 1] \times [0, 1].$$

Здесь  $U = [u^3 \ u^2 \ u \ 1]$ ,  $V = [v^3 \ v^2 \ v \ 1]$ ,  $M_b = \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$  – матрица Безье, а

$$P = \begin{pmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \\ P_{41} & P_{42} & P_{43} & P_{44} \end{pmatrix} \text{ – набор управляющих точек, см. рис. 10.19.}$$

Рис. 10.19 хорошо иллюстрирует пример важного свойства куска Безье: все токи куска Безье лежат внутри выпуклой оболочки, натянутой на 16 управляющих точек. Иногда эту оболочку называют *управляющим многогранником* для куска Безье.

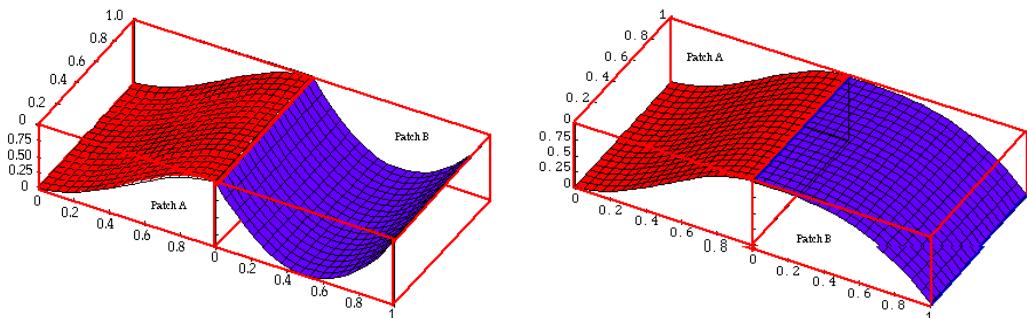


Рис. 10.20. Примеры сшивки кусков Безье

Пусть два куска Безье определяются своими управляющими наборами точек:

$$\text{кусок 1} - \begin{pmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \\ P_{41} & P_{42} & P_{43} & P_{44} \end{pmatrix}, \text{кусок 2} - \begin{pmatrix} P'_{11} & P'_{12} & P'_{13} & P'_{14} \\ P'_{21} & P'_{22} & P'_{23} & P'_{24} \\ P'_{31} & P'_{32} & P'_{33} & P'_{34} \\ P'_{41} & P'_{42} & P'_{43} & P'_{44} \end{pmatrix}$$

*Упражнение.* Совершенно очевидно как производить сшивку двух кусков для обеспечения непрерывности при переходе от куска к куску, рис. 10.20, слева. Для сшивки двух кусков вдоль оси X, обеспечивающей непрерывность, как показано справа на рис. 10.20, необходимо выполнить следующие условия:

$$\begin{aligned} P_{14} - P_{13} &= k(P'_{14} - P'_{13}) \\ P_{24} - P_{23} &= k(P'_{24} - P'_{23}), \quad k > 0. \\ P_{34} - P_{33} &= k(P'_{34} - P'_{33}) \\ P_{44} - P_{43} &= k(P'_{44} - P'_{43}) \end{aligned}$$

Записать требуемые условия для сшивки по оси Y.

## 10.8. Несколько часто используемых форм

*Сpirаль* (или винтовая линия) с радиусом  $a$  и шагом  $b$ :  $r(t) = \begin{pmatrix} a \cos(t) \\ a \sin(t) \\ bt \end{pmatrix}$ .

*Сфера* радиуса  $a$ :  $r(\theta, \varphi) = \begin{pmatrix} a \sin(\varphi) \cos(\theta) \\ a \sin(\varphi) \sin(\theta) \\ a \cos(\varphi) \end{pmatrix}, 0 \leq \theta \leq 2\pi, -\pi \leq \varphi \leq \pi$ .

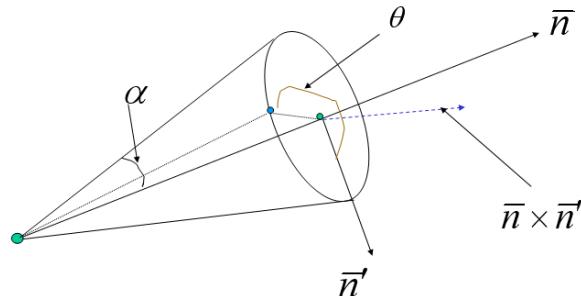


Рис. 10.21. Параметризация конуса

*Конус* с углом  $\alpha$  при вершине:  $r(l, \theta) = l \cdot \bar{n} + l \cdot \operatorname{tg}(\alpha) \cdot \cos(\theta) \cdot \bar{n}' + l \cdot \operatorname{tg}(\alpha) \cdot \sin(\theta) \cdot [\bar{n} \times \bar{n}']$ , см. рис. 10.21. Формула позволяет получить координаты точек поверхности конуса по:  $l$  – расстоянию от вершины конуса по оси  $\bar{n}$ ;  $\theta$  – углу от некоторого направления  $\bar{n}'$  в плоскости перпендикулярной  $\bar{n}$ .

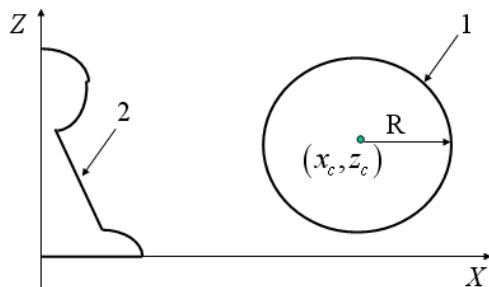


Рис. 10.22. Примеры образующих

*Поверхности вращения.* Это очень широкий класс поверхностей, они легко записываются в параметрической форме. Как правило, образующая представляется в виде параметрической

кривой на плоскости  $XZ$  и имеет вид:  $r_{xz}(t) = \begin{pmatrix} x(t) \\ 0 \\ z(t) \end{pmatrix}$ , а сама поверхность вращения

представляется формулой:  $r(t, \theta) = \begin{pmatrix} x(t) \cdot \cos \theta \\ x(t) \cdot \sin \theta \\ z(t) \end{pmatrix}, 0 \leq \varphi \leq 2\pi$ . Можно вырезать только сектор

поверхности, тогда берется  $0 \leq \varphi \leq \alpha$ . На рис. 10.22 даны два примера образующих:

1. Кривая 1 – это окружность радиуса  $R$  с центром в точке  $(x_c, 0, z_c)$ . Очевидно, что полученная поверхность вращения будет тором.
2. Кривая 2 – образующая шахматной пешки.

*Упражнение.* Записать сферу, цилиндр и конус как фигуры вращения.

В качестве примера применения параметрических поверхностей на практике рассмотрим задачу вычисления пути фрезы для получения поверхности параболоида вращения,

определенного формулой:  $r(t, \varphi) = \begin{pmatrix} 2at \cos \varphi \\ 2at \sin \varphi \\ at^2 \end{pmatrix}$ . Определим вектор перпендикулярный к

$$\text{поверхности в точке } (t, \varphi): \bar{p} = \frac{\partial r}{\partial t} \times \frac{\partial r}{\partial \varphi} = \begin{vmatrix} \bar{i} & \bar{j} & \bar{k} \\ 2a \cos \varphi & 2a \sin \varphi & 2at \\ -2a \sin \varphi & 2a \cos \varphi & 0 \end{vmatrix} = \begin{pmatrix} -4a^2 t \cos \varphi \\ -4a^2 t \sin \varphi \\ 4a^2 t \end{pmatrix}.$$

$$|\bar{p}|^2 = 16a^4 t^2 (1 + t^2), \text{ тогда нормаль будет выражаться формулой: } \bar{n} = \frac{1}{\sqrt{1+t^2}} \begin{pmatrix} -t \cos \varphi \\ -t \sin \varphi \\ 1 \end{pmatrix}.$$

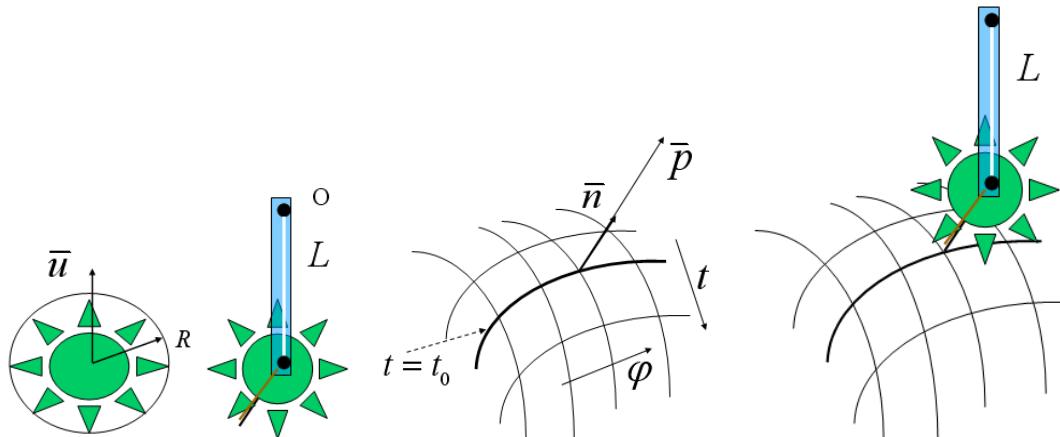


Рис. 10.22. К задаче о пути фрезы

Пусть фреза такова, как показано на рис. 10.22, здесь: фреза имеет сферическую форму с радиусом  $R$  и крепится на ручке длиной  $L$ ; точка  $O$  – точка соединения с управляющим механизмом;  $\bar{u}$  – единичный вектор, направленный вдоль ручки. Требуется написать формулу пути для точки  $O$ , чтобы фреза прошла по поверхности по параметрической линии,

имеющей уравнение  $t = t_0$ . Итак, уравнение параметрической линии  $r(t_0, \varphi) = \begin{pmatrix} 2at_0 \cos \varphi \\ 2at_0 \sin \varphi \\ at_0^2 \end{pmatrix}$ ,

нормаль в точке  $r(t_0, \varphi)$ :  $\bar{n}(\varphi) = \frac{1}{\sqrt{1+t_0^2}} \begin{pmatrix} -t_0 \cos \varphi \\ -t_0 \sin \varphi \\ 1 \end{pmatrix}$ . Окончательно искомый путь точки  $O$  описывается уравнением:  $r_O(t_0, \varphi) = r(t_0, \varphi) + R\bar{n} + L\bar{u}$ .

## 10.9. Некоторые необычные формы

Геометрическое моделирование предусматривает этап создания примитивов, т.е. определенных базисных геометрических форм, из которых пользователь будет конструировать более крупные детали или целые сцены. Различные справочники содержат множество форм, описываемых аналитически. Это позволяет привлечь различные куски этих поверхностей, нарезая из них куски Эрмита с верными касательными. В дальнейшем их можно конвертировать в куски Безье и продолжить совершенствование формы к требуемому виду, применяя интерактивный режим. В качестве примера можно вспомнить суперэллипсы, введенные Фланаганом еще в 1867 году. Сверхэллипс на плоскости описывается

уравнением:  $\frac{x^n}{a^n} + \frac{y^n}{b^n} = 1$ . При больших  $n$  его форма приближается к прямоугольнику, но сохраняется гладкость в углах. Интересен подход Пентланда, который исследовал применение *суперквадриков*. Он несколько обобщил параметрическое представление сферы и определил суперквадрику следующим параметрическим заданием:

$$p(s, t) = \begin{pmatrix} x(s, t) \\ y(s, t) \\ z(s, t) \end{pmatrix} = \begin{pmatrix} C_t^{e_1} \cdot C_s^{e_2} \\ C_t^{e_1} \cdot S_s^{e_2} \\ S_t^{e_1} \end{pmatrix}, \quad -\frac{\pi}{2} \leq t \leq \frac{\pi}{2}, 0 \leq s \leq \pi,$$

здесь:  $e_1$  и  $e_2$  – экспоненциальные параметры формы,  $C_t = \cos t$ ,  $S_t = \sin t$ ,  $C_s = \cos s$ ,  $S_s = \sin s$ . Во избежание проблем с вычислениями будем считать, что:

$$(\cos t)^e = (|\cos t|)^e \cdot \text{sign}(\cos t).$$

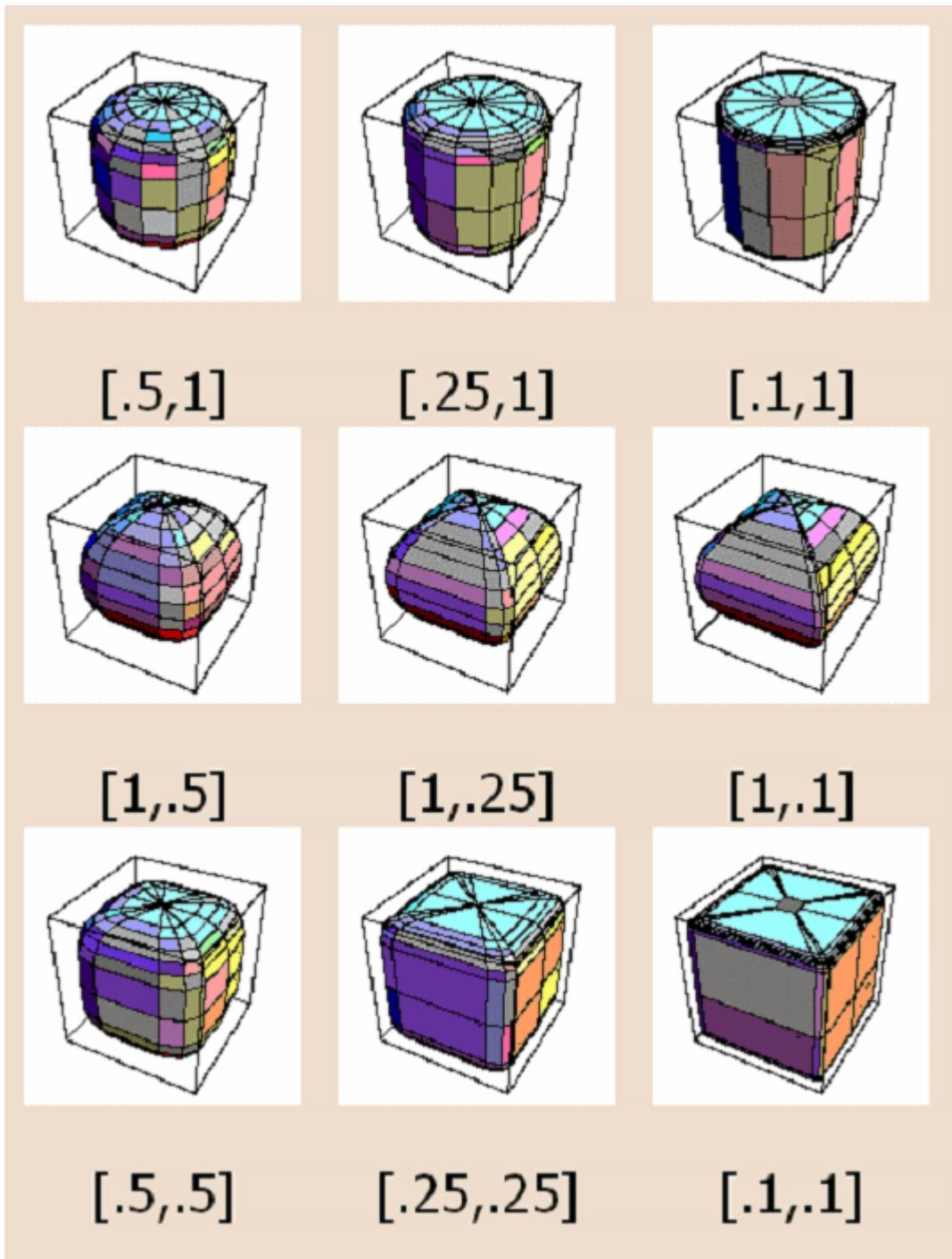


Рис. 10.23. Параметры формы  $e_1, e_2$  не больше 1

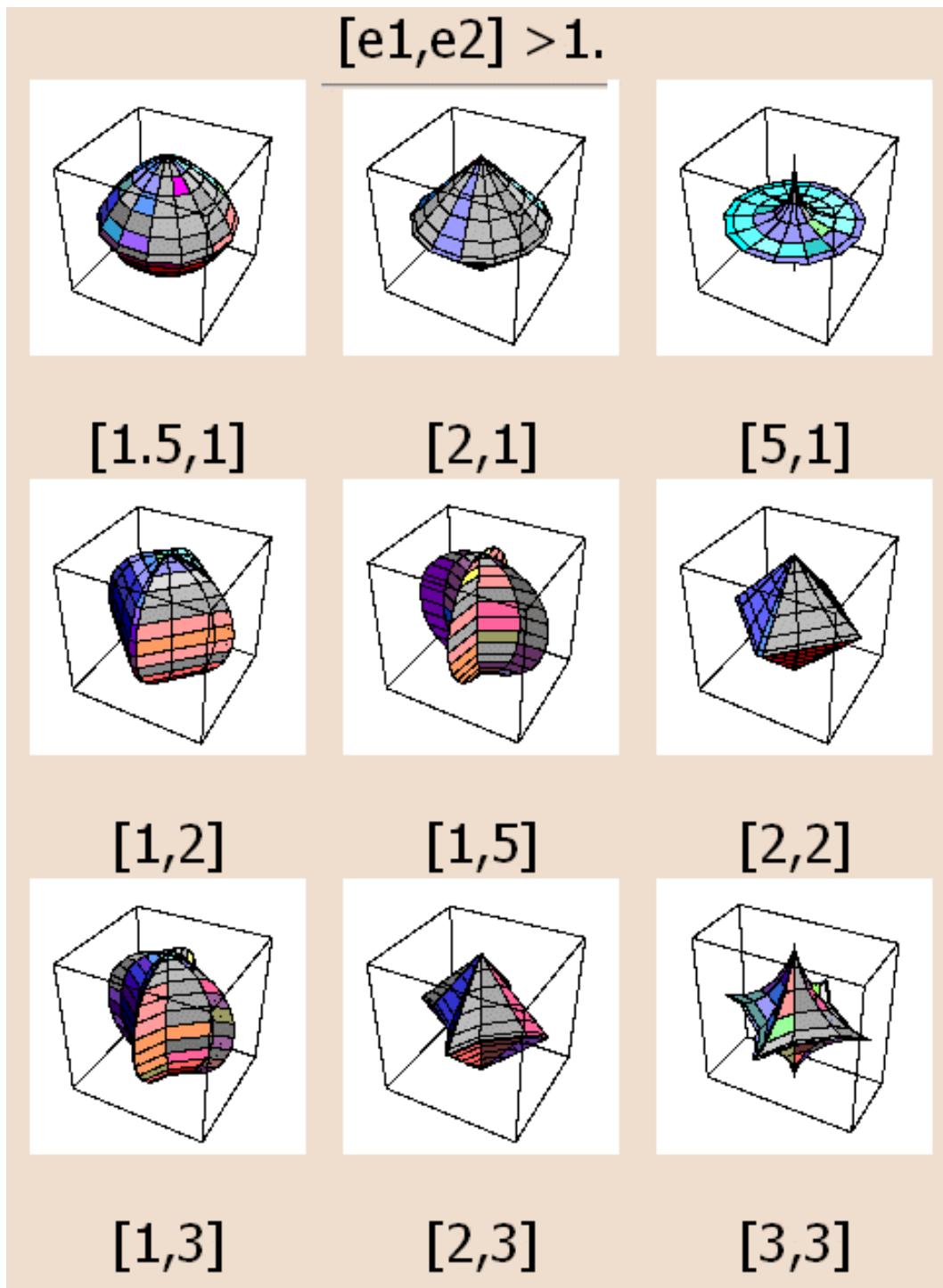


Рис. 10.24. Параметры формы  $e_1, e_2$  больше равны 1

Для более гибкого управления формой вводятся независимые положительные масштабные множители по осям:

$$p(s, t) = \begin{pmatrix} x(s, t) \\ y(s, t) \\ z(s, t) \end{pmatrix} = \begin{pmatrix} a_x \cdot C_t^{e_1} \cdot C_s^{e_2} \\ a_y \cdot C_t^{e_1} \cdot S_s^{e_2} \\ a_z \cdot S_t^{e_1} \end{pmatrix}. \quad (10.15)$$

Теперь мы можем нарезать куски из суперкуадрика. Эти куски можно использовать прямо с их параметризацией (10.15), поскольку также аналитически выражается нормаль в точке  $p(s, t)$ :

$$N(s, t) = \begin{pmatrix} \frac{1}{a_x} C_t^{2-e_1} \cdot C_s^{2-e_2} \\ \frac{1}{a_y} C_t^{2-e_1} \cdot S_s^{2-e_2} \\ \frac{1}{a_z} S_t^{2-e_1} \end{pmatrix}.$$

Так вот Пентланд, опираясь только на такие куски, использовал только 13 примитивов, чтобы построить приемлемую модель человеческой головы.

## 10.10. Полигональные модели

В предыдущих разделах мы рассматривали различные способы, которые позволяют строить определенные базовые элементы геометрии сцены. Но в компьютерной графике предпочтительнее, когда геометрия сцены задается в виде набора плоских многоугольников. Итак, рассмотрим все примитивы сцены определенные нами в параметрическом виде  $r(u, v)$ ,  $(u, v) \in [a, b] \times [c, d]$ .

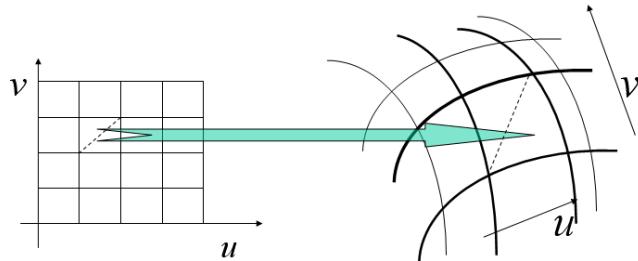


Рис. 10.25. К вопросу о полигонизации

Как правило, на такую поверхность наносится сетка параметрических линий:

$$r(u_i, v), u_1 = a, u_2, \dots, u_n = b; v \in [c, d] \text{ и } r(u, v_j), u \in [a, b]; v_1 = c, v_2, \dots, v_m = d.$$

Эта сетка образует на поверхности множество криволинейных четырехугольников. Соединяя четыре вершины отрезками прямых. Если полученный четырехугольник не планарный, то на основе четырех вершин строим два треугольника. Всегда стоит вопрос: "Как выбрать шаг сетки в области изменения параметров?". Данный вопрос решается по-разному в зависимости от того, для чего полигональная поверхность будет применяться. Известны следующие подходы:

1. Выбор шага в интерактивном режиме, т.е. пользователь определяет качество полученной полигональной поверхности "на глаз".
2. На основе сравнения длин сторон криволинейного четырехугольника или треугольника с соответствующими длинами аппроксимирующего многоугольника. Если относительная погрешность выходит из заданных пределов, то осуществляется измельчение шага.
3. На основе сравнения площадей криволинейного четырехугольника или треугольника с площадью аппроксимирующего многоугольника. Если относительная погрешность выходит из заданных пределов, то осуществляется измельчение шага. Формулу вычисления площади криволинейного четырехугольника возьмем из книги [18]:  

$$S_{ij} = \iint_{d_{ij}} |r_u \times r_v| du dv, \text{ где } d_{ij} = [u_i, u_{i+1}] \times [v_j, v_{j+1}].$$

Последние два подхода достаточно эффективно применять в случаях, когда известно аналитическое представление поверхности.

*Упражнение.* Получить полигональную модель тора с заданной погрешностью.

## 10.11. Другие преобразования формы

В предыдущих разделах мы рассматривали различные способы, которые позволяют строить определенные базовые элементы геометрии сцены. Перед тем, как разместить эти элементы в сцене, их можно подвергнуть какому-либо дополнительному преобразованию и расширить таким образом арсенал форм. Как правило, эти преобразования применяются не к исходным, а к полигональным моделям. Рассмотрим два популярных преобразования. Они достаточно просты и могут служить в качестве методического приема для разработки пользовательских преобразований формы.

### 10.11.1. Сведение на конус – Z-taper

Вначале требуется выровнять объект вдоль оси Z, а затем применить масштабирующую функцию к каждой точке объекта. Линейный вариант преобразования:

$$x' = (kz + l)x,$$

$$y' = (kz + l)y,$$

$$z' = z.$$

Если его применить к цилиндру, то получится конус или усеченный конус. В более общем виде преобразование записывается так:

$$x' = f(z)x,$$

$$y' = f(z)y,$$

$$z' = z.$$

### 10.11.2. Скручивание – Z-twist

Аналогично предыдущему объект сначала выравнивается по отношению к оси Z, затем вращаем точки поверхности вокруг оси Z на угол, являющийся функцией z :

$$\theta = f(z),$$

$$x' = x \cos(\theta) - y \sin(\theta),$$

$$y' = x \sin(\theta) + y \cos(\theta),$$

$$z' = z.$$

Это преобразование можно применять совместно с предыдущим.

*Упражнение.* Предложить одно-два преобразования формы.

### 10.11.3. Экструзия или sweeping

Достаточно наглядным примером экструзии является тюбик с пастой. Горловина тюбика имеет в сечении некоторую плоскую фигуру  $S$ , как правило, односвязную, т.е. ее граница состоит из одного контура, например, многоугольника (см. рис. 10.26). Пусть дана кривая в пространстве  $r(t), t \in [a, b]$ . Будем двигать тюбик так, чтобы центр горловины находился на кривой  $r(t)$ , а сама горловина (плоскость фигуры  $S$ ) была всегда перпендикулярна кривой. На рис. 10.26, справа приведен плоский вариант экструзии, когда происходит "заметание" (sweeping) точек плоскости отрезком,двигающимся по кривой.

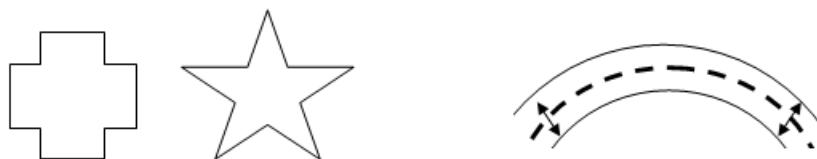


Рис. 10.26. Два примера сечения тела (слева), пример плоской экструзии (справа)

Данную операцию можно усложнить, если фигура сечения  $S$  будет поворачиваться относительно своего центра на угол, зависящий от параметра  $t$  – аналог операции Z-twist. Аналогично можно потребовать, чтобы фигура сечения  $S$  масштабировалась аналогично операции Z-taper в зависимости от параметра  $t$ .

## 10.12. Представления объектов

До этого момента мы рассматривали только поверхности, но машиностроительные детали в САПР, сцены в системах визуализации и т.д. составляются, как правило, из телесных объектов (solids). Конструирование тел (solid modeling) опирается на следующие основные представления объектов:

### 10.12.1. Границное представление

Все рассмотренные выше методы позволяют построить куски поверхностей. Если эти поверхности образуют границу тела, то говорят о *границном представлении* тела (boundary representation или кратко B-reps).

### 10.12.2. Воксельное представление

Аналогично растрю на плоскости, разбитому на одинаковые квадратные пиксели, пространство  $R^3$  разбивается на одинаковые кубики, которые по аналогии называются вокселями (voxel – volume element) – элементами объема. Ряд задач проектирования легче решать на воксельных представлениях: расчет объема, расчет момента инерции и т.д.

### 10.12.3. Конструктивная геометрия

Конструктивная геометрия тел (constructive solid geometry – CSG) может рассматриваться с двух точек зрения. Во-первых, CSG – это способ построения тела. Во-вторых, это способ описания объекта.

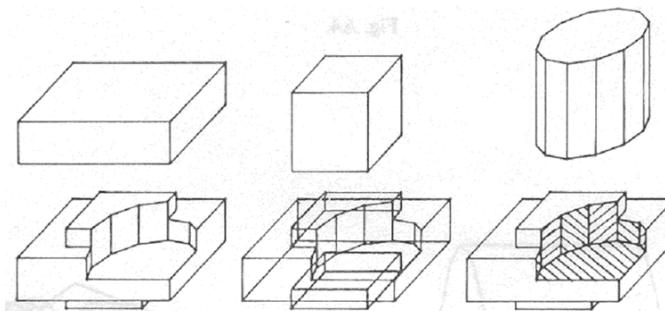


Рис. 10.27. Конструктивное построение объекта

Большинство систем, предоставляющих средства CSG, базируются на регуляризованных теоретико-множественных операциях (РТМО) и определенном наборе *базовых элементов формы* (БЭФ). Множество БЭФ состоит из элементарных неделимых форм типа бокс, шар, конус, и т.д. Множество операций может включать, кроме РТМО, аффинные преобразования, Z-taper, Z-twist, Sweeping и др. В целом множество БЭФ и множество операций определяют разнообразие получаемых форм. Если рассмотреть историю построения объекта, то она описывается иерархической структурой, на рис. 10.28 приводится структура для объекта, изображенного на рис. 10.27.

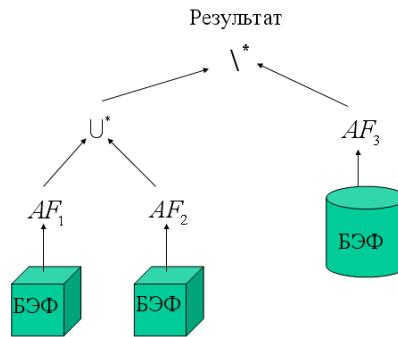


Рис. 10.28. История конструктивного построения объекта

Здесь использованы два разных БЭФ, операции РТМО  $\setminus^*$  и  $\cup^*$ , три аффинных преобразования.

В большинстве систем объект хранится в виде аналогичной структуры данных, отражающей конструктивную историю его создания.

#### 10.12.4. Функциональное представление

Толчок этому направлению в представлении геометрических объектов дали R-функции, введенные Рвачевым. В дальнейшем это представление было развито в сторону геометрического моделирования и часто называется *функциональным представлением тел* (functional representation – F-reps) [17]. Рассмотрим основную идею,ложенную в его основу.

Пусть у нас имеется геометрический объект  $O$  и функция  $f_o(x, y, z)$ , такая что

$$f_o(x, y, z) = \begin{cases} > 0, & (x, y, z) \in O^\circ \\ = 0, & (x, y, z) \in \partial O \\ < 0, & (x, y, z) \notin (O^\circ \cup \partial O) \end{cases}.$$

И пусть аналогичные функции  $f_1$  и  $f_2$  существуют для объектов  $O_1$  и  $O_2$  соответственно. Оказывается, что для объекта, являющимся результатом теоретико-множественной операции над этими объектами, можно построить аналогичную функцию.

1. Объединение объектов  $O_3 = O_1 \cup O_2 \Rightarrow f_3 = f_1 + f_2 + \sqrt{f_1^2 + f_2^2}$ .
2. Пересечение объектов  $O_3 = O_1 \cap O_2 \Rightarrow f_3 = f_1 + f_2 - \sqrt{f_1^2 + f_2^2}$ .
3. Дополнение объектов  $O_3 = -O_1 = R^3 \setminus O_1 \Rightarrow f_3 = -f_1$ .
4. Разность объектов  $O_3 = O_1 \setminus O_2 = O_1 \cap -O_2$ .

Здесь мы рассмотрели простейший вариант функционального задания геометрических объектов. Очевидно, что это представление хорошо согласуется с CSG. На рис. 10.29 даны примеры объектов, которые рассчитывались на основе F-reps.

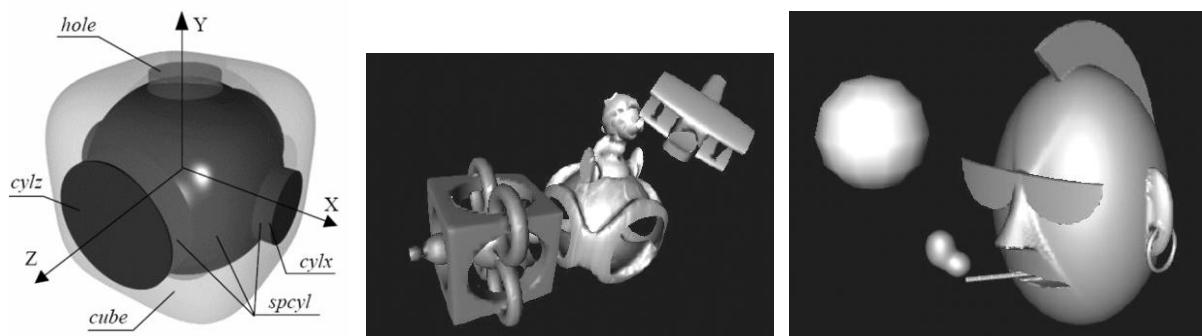


Рис. 10.29. Примеры сцен, рассчитанных по F-reps [17]

*Упражнение.* На рис. 10.30 даны два объекта: 1) круг радиуса 1.5; 2) эллипс с полуосями 1.0 и 2.0. Построить функции для объединения и пересечения этих объектов.

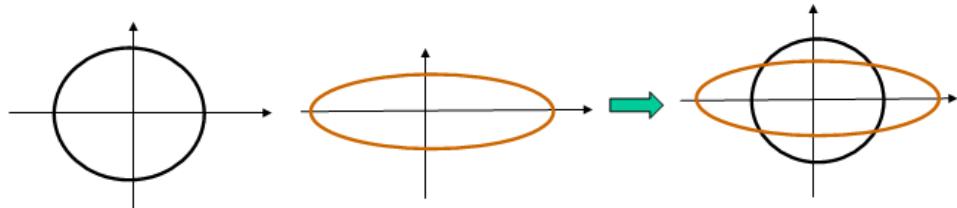


Рис. 10.30. К упражнению по F-reps

### 10.13. Заключение

В данном разделе мы рассмотрели ряд методов построения геометрической формы объектов и некоторые из операций, позволяющие проводить геометрические модификации уже построенных форм. Здесь мы не рассмотрели аффинные преобразования, которые скорее применяются для сборки сцен, чем для создания базовых форм. Отметим, что рассмотренный материал дает достаточно представительный вид о данной области, по крайней мере, для того, чтобы строить достаточно сложные сцены.

## 11. Преобразования координат

### 11.1. Введение

С преобразованиями координат мы сталкиваемся на различных этапах решения задачи визуализации пространственных сцен. Ряд авторов, например, Экберт (Использовались материалы курса 15-462 (Heckbert) <http://www-2.cs.cmu.edu/afs/cs/academic/class/15462/web/notes/transform.ppt.gz>) делит их на следующие группы: модельные, видовые и анимационные.

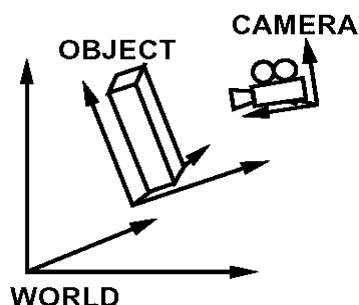


Рис. 11.1. Системы координат процесса рендеринга

Аналогично подразделяются и системы координат (СК): модельные или объектные (object) координаты – СК, в которых задаются базовые формы объектов; мировые (world) координаты или СК сцены, в которой собирается вся геометрия сцены; СК камеры.

### 11.1.1. Модельные координаты и преобразования

При конструировании (задании) трехмерных сцен осознанно или неосознанно пользователь выполняет сначала анализ, когда он всю сцену разбивает на отдельные части, более сложные части разбивает на более мелкие части и т.д. В конце концов, он получает набор базовых геометрических форм, из которых состоит сцена. Затем производится синтез сцены, когда из мелких деталей собираются более крупные, из тех еще более крупные, и, наконец, собирается вся сцена.

Каждый объект задается в удобной системе координат, т.е. в такой системе координат, в которой расчет опорных точек объекта легче или нагляднее. Для примера рассмотрим задание куба в разных системах координат, рис. 11.2.

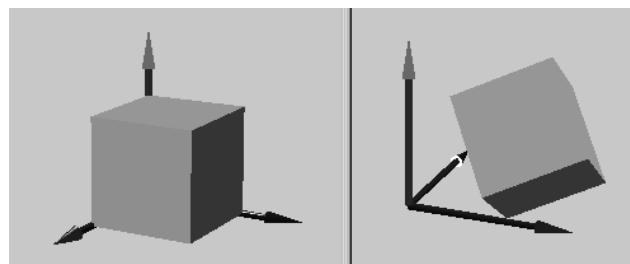


Рис. 11.2. Удобная СК (слева), неудобная СК (справа)

Система координат, в которой задается объект, называется *модельной* или *объектной*, а координаты – *модельными*. Координатные преобразования, которые обеспечивают размещение объекта в модельной СК более сложного объекта, называются *модельными* преобразованиями. Таким образом, модельные преобразования служат: а) для создания сложных моделей из простых путем позиционирования, вращения и масштабирования; б) для преобразования объектных координат в мировые. Отметим, что возможность масштабирования облегчает компьютерное моделирование по сравнению с натурным моделированием (конструктор LEGO), поскольку требует значительно меньшего числа базовых форм.

### 11.1.2. Видовое преобразование

Видовым преобразованием называется преобразование из мировой системы координат в систему координат изображения (вида). Фактически это преобразование состоит из двух: 1) перевод координат из мировой СК в СК камеры; 2) преобразование проецирования на плоскость изображения. Другими словами, оно строится на основе положения виртуальной камеры в мире сцены и спецификаций самой камеры. Часто говорят о преобразовании порта вывода (viewport transformation), которое служит для определения видимой части мира в окне на экране, рис. 11.3. Часто, по умолчанию, порт вывода – это вся клиентская область экранного окна.

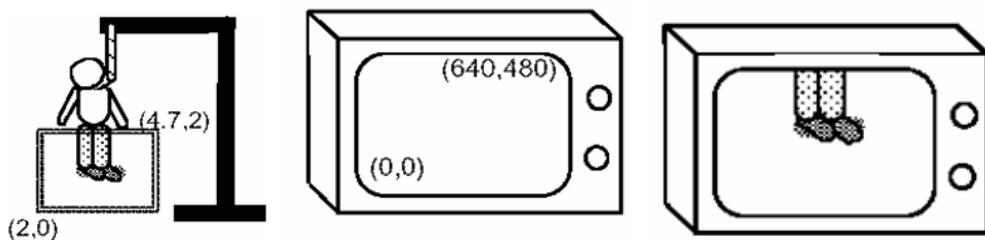


Рис. 11.3. Порт вывода

### 11.1.3. Анимационные преобразования

Анимации – это динамические изображения. Анимационные преобразования определяют варьирование координат объектов сцены во времени для создания движений и изменения форм объектов сцены. В данном курсе они не будут рассматриваться.

### 11.1.4. Сведения из векторной алгебры

Предполагается, что следующие понятия и факты из векторной алгебры известны читателю:

- Линейная комбинация векторов.
  - Линейная независимость векторов.
  - Базис, ортонормированный базис.
  - Компоненты или координаты вектора.
  - Вектор в базисе представляется однозначно. При смене базиса вектор не меняется, меняются его компоненты.
  - Преобразование  $F$  называется *линейным*, если для любых векторов  $a, b$  и числа  $k$  следует  $F(a+b) = F(a) + F(b), F(ka) = kF(a)$ .
  - Преобразование  $A$  называется *аффинным*, если оно линейное плюс сдвиг на определенный вектор  $b$ :  $A(a) = F(a) + b$ .
  - Любое линейное преобразование полностью специфицируется его действием на базисные векторы. Применяются матрицы для надежности:
    - Матрица  $n \times n$  представляет линейную функцию в  $n$ -мерном пространстве.
    - $i$ -й столбец показывает как она действует на соответствующий вектор базиса.
    - Преобразование – это линейная комбинация столбцов матрицы.
    - Обычно вычисляют по другому: скалярно множат строку  $i$  на исходный вектор, чтобы получить компоненту  $i$  выходного вектора:
- $$V' = F(V) \Rightarrow \begin{pmatrix} v'_1 \\ v'_2 \\ v'_3 \end{pmatrix} = \begin{pmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}.$$

## 11.2. Основные 2D преобразования

При формировании двумерных сцен используются 3 основных преобразования координат на плоскости:

- Сдвиг (Translation) на вектор  $t$ .  $v' = v + t$  или в координатном виде:  $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$ .
- Масштабирование (Scale) по осям координат.  $v' = Sv$  или  $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$ .
- Поворот (Rotation) на угол  $\theta$ .  $v' = R(\theta)v$  или  $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$ .

В компьютерной графике основное внимание уделяется геометрическим формам, которые полностью определяются некоторым количеством опорных точек. К ним относятся все многоугольники, ломаные, кривые Безье. Действительно, для преобразования

многоугольника достаточно преобразовать только его вершины. Таким образом, все рассмотрения можно делать для отдельной точки. В течение формирования сцены элементарная форма подвергается преобразованиям несколько раз, например, круг входит в деталь – преобразование  $T_1$ , деталь в более крупный узел – преобразование  $T_2$ , а уже узел встраивается в механизм – преобразование  $T_3$ . Масштабирование и поворот являются линейными преобразованиями. Очевидно, что любую последовательность из этих двух преобразований можно собрать в единую матрицу. И вместо последовательного перемножения на соответствующие матрицы нам достаточно сделать только одно. А вот сдвиг – это аффинное преобразование, оно в матричном виде не представляется, поэтому в приведенном примере придется для каждой опорной точки последовательно применять все три преобразования:  $T_1$ ,  $T_2$ ,  $T_3$ . Рассмотрим показательный пример (рис. 11.4) поворота объекта-топора вокруг точки  $P$ . Поворот выполняется в три шага:

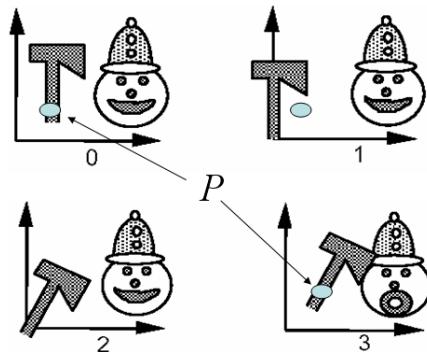


Рис. 11.4. Поворот вокруг точки

- Перенос объекта в начало координат (сдвиг).
- Поворот вокруг начала координат на требуемый угол (поворот).
- Обратный перенос объекта на место (сдвиг).

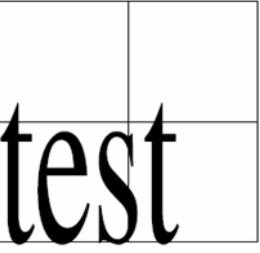
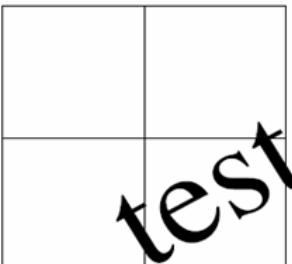
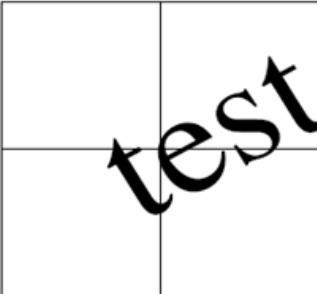
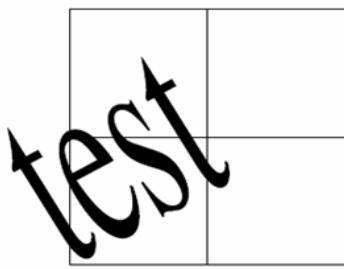
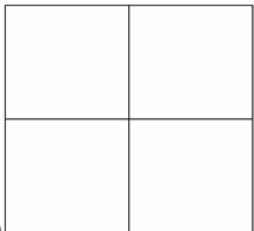
В чем основное неудобство? В том, что мы должны будем сохранять всю последовательность элементарных преобразований координат и при необходимости отрабатывать их также по одному.

### 11.3. 2D преобразования в Postscript

Известно, что Postscript – это язык лазерного принтера. Это достаточно универсальный язык программирования, основанный на стековом подходе: вместо "a + c" пишем "a c +". Серия картинок в табл. 11.1 показывает применение преобразований в рисовании.

Таблица 11.1. Рисование в Postscript

|                                        |                                            |                                          |
|----------------------------------------|--------------------------------------------|------------------------------------------|
| 0 0 moveto<br>(test) show              | 1 0 translate<br>0 0 moveto<br>(test) show | 30 rotate<br>0 0 moveto<br>(test) show   |
|                                        |                                            |                                          |
| 1 2 scale<br>0 0 moveto<br>(test) show | 1 0 translate<br>30 rotate<br>0 0 moveto   | 30 rotate<br>1 0 translate<br>0 0 moveto |

|                                                                                   | (test) show                                                                       | (test) show                                                                         |
|-----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
|  |  |  |
| 30 rotate<br>1 2 scale<br>0 0 moveto<br>(test) show                               | 30 rotate<br>1 2 scale<br>0 0 moveto<br>(test) show                               | -1 1 scale<br>0 0 moveto<br>(test) show                                             |
|  |  |  |

Из этих нескольких примеров видно, что операция сдвига встречается достаточно часто, чтобы ее можно было бы игнорировать.

#### 11.4. Однородные координаты и 2D преобразования

Итак, надо постараться представить сдвиг в матричном виде. Давайте сделаем следующий "трюк" – добавим по лишней координате и запишем следующее выражение:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

Действительно это выражение позволяет правильно выполнить сдвиг  $\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ 0 \end{pmatrix}$ .

Сопоставим каждой точке плоскости  $P = \begin{pmatrix} x \\ y \end{pmatrix}$  следующий столбец из трех чисел  $\begin{pmatrix} X \\ Y \\ W \end{pmatrix}$ . Мы

будем говорить, что он определяет точку  $P$  для любого значения  $W > 0$ , если выполняются условия:  $x = \frac{X}{W}$  и  $y = \frac{Y}{W}$ . В случае  $W = 0$  получаем точку на бесконечности – это *вектор направления*  $\begin{pmatrix} X \\ Y \end{pmatrix}$ .  $W$  называется *однородной координатой*. А такие матрицы определяют

однородные преобразования. Поскольку пока никакой необходимости нет использовать  $W \neq 1$ , то мы будем механически добавлять к точке  $W = 1$  перед операцией, а после операции просто отбрасывать.

Оказывается, что  $\begin{pmatrix} X \\ Y \\ W \end{pmatrix}$  – это координаты из 3D проективного пространства, и все ненулевые

кратные  $W \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$  определяют одну и ту же точку в декартовых координатах –  $\begin{pmatrix} x \\ y \end{pmatrix}$ .

Наконец, запишем все наши преобразования в матричном виде для применения в однородных координатах:

- Сдвиг –  $T(t_x, t_y) = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$ .
- Поворот –  $R(\theta) = \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$ .
- Масштабирование –  $S(s_x, s_y) = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$ .

Теперь любая последовательность сдвигов / вращений / масштабирований на плоскости может быть выражена одной матрицей. Обратимся снова к примеру, приведенному на рис. 11.3. Полное преобразование любой точки объекта может быть записано одной матрицей:  $M = T(-P) \cdot R(\theta) \cdot T(P)$ .

## 11.5. Правосторонние и левосторонние системы координат

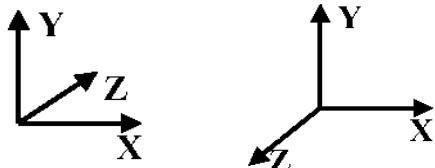


Рис. 11.5. Левосторонняя (слева) и правосторонняя (справа) СК

Переходя от плоскости к пространству, к имеющимся двум координатам  $x$  и  $y$  необходимо добавить еще одну декартову координату  $z$ . Соответствующая ось  $Z$  определяется через оси  $X$  и  $Y$  на основе векторного произведения, которое определяется правилом левой или правой руки, что и дало название системе, см. рис. 11.5, 11.6.

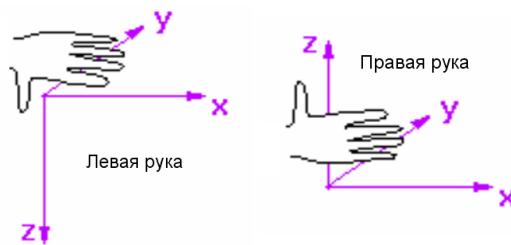


Рис. 11.6. Определение направления оси  $Z$

Для левосторонней системы координат выбрано соотношение для ортов:  $\vec{Y} \times \vec{X} = \vec{Z}$ , а для правосторонней –  $\vec{X} \times \vec{Y} = \vec{Z}$ .

*Замечание.* При программировании модулей надо точно знать, в какой из этих СК представлены данные. Игнорирование этой информации может привести к неожиданным артефактам на изображении.

В отличие от плоскости в пространстве есть понятие оси вращения. Направление вращения зависит от выбранной системы координат (левой или правой). Положительное вращение, т.е. на положительный угол (верно для обеих систем), определяется следующим образом:

- Вокруг оси  $X$  : от  $+Y$  к  $+Z$ .
- Вокруг оси  $Y$  : от  $+Z$  к  $+X$ .
- Вокруг оси  $Z$  : от  $+X$  к  $+Y$ .

Заметим, что на плоскости положительное вращение осуществляется против часовой стрелки, что соответствует правосторонней СК.

Существует еще один момент, на который надо обращать внимание при чтении документации или разборе алгоритмов, – какое соглашение о координатной записи векторов или точек принято автором: вектор-столбец или вектор строки. Соответственно меняется и порядок матричной записи координатных преобразований. Для случая вектора-столбца мы пишем

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix},$$

и последовательность преобразований с матрицами  $A$ ,  $B$ ,  $C$  и  $D$  записывается как  $p' = ABCDp$ . Для случая вектора-строки:

$$(x' \ y' \ 1) = (x \ y \ 1) \begin{pmatrix} m_{11} & m_{21} & m_{31} \\ m_{12} & m_{22} & m_{32} \\ m_{13} & m_{23} & m_{33} \end{pmatrix} \text{ и } p'^T = p^T D^T C^T B^T A^T,$$

соответственно. Здесь  $(\cdot)^T$  – операция транспонирования. Далее мы будем использовать правостороннюю СК без дополнительных упоминаний, кроме случаев, когда это может привести к двусмысленности. Отметим, что в документации (например, по Direct3D) часто используется левосторонняя СК и запись векторов в виде строк.

## 11.6. Однородные преобразования в пространстве

Аналогично плоскому случаю в записи точки  $p = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$  добавляем однородную координату

$W \neq 0$  и получаем  $p = \begin{pmatrix} Wx \\ Wy \\ Wz \\ W \end{pmatrix}$ . До тех пор, пока нам это не потребуется мы используем  $W = 1$ .

Определим следующие однородные преобразования:

- Сдвиг на вектор –  $T(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$ .
- Масштабирование по осям –  $S(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ .
- Поворот вокруг оси  $X$  на угол  $\theta_x$  –  $R_x(\theta_x) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta_x & -\sin \theta_x & 0 \\ 0 & \sin \theta_x & \cos \theta_x & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ .
- Поворот вокруг оси  $Y$  на угол  $\theta_y$  –  $R_y(\theta_y) = \begin{pmatrix} \cos \theta_y & 0 & \sin \theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta_y & 0 & \cos \theta_y & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ .
- Поворот вокруг оси  $Z$  на угол  $\theta_z$  –  $R_z(\theta_z) = \begin{pmatrix} \cos \theta_z & -\sin \theta_z & 0 & 0 \\ \sin \theta_z & \cos \theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ .

Легко можно показать, что любая последовательность сдвигов / вращений / масштабирований в пространстве может быть выражена одной матрицей. Выше представлены матрицы преобразований для правосторонней системы координат, для левосторонней СК их надо транспонировать. Отметим, что при применении этих преобразований к отрезкам, граням и т.д. достаточно преобразовывать только опорные вершины.

В пространстве появляется понятие *оси вращения*, которая может определяться любым вектором, проходящим через начало СК. Известно, что любой поворот – это комбинация поворотов вокруг координатных осей на углы  $\theta_x, \theta_y, \theta_z$ , которые известны под названием углов Эйлера. Но представление произвольного поворота через углы Эйлера не однозначно, что делает проблематичным опираться на них при построении анимаций.

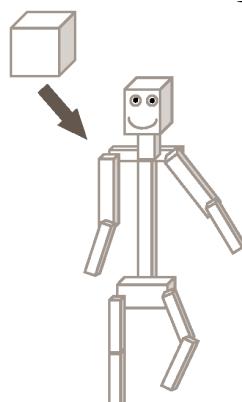


Рис. 11.7. Применение аффинных моделирующих преобразований

Аффинные преобразования являются модельными, т.к. обеспечивают построение сложных объектов из базовых элементов формы, например, на рис. 11.7 показано как из единственного БЭФа – кубика – построен сложный объект. Если мы заставим его ходить, то нам придется применить вращения и, тогда необходимо искать замену углам Эйлера поскольку их применение приводит нередко к дерганым анимациям. Конечно, при помощи данных преобразований мы не сможем моделировать *физически корректные* деформации, т.е. сокращения мышц и т.п.

Часто применяется термин *преобразования твердого тела*, который определяет преобразования без деформаций объекта и является комбинацией любого числа поворотов и сдвигов. В общем виде они представляются матрицей

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} & t_x \\ m_{21} & m_{22} & m_{23} & t_y \\ m_{31} & m_{32} & m_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

где подматрица 3x3 – это матрица вращения.

## 11.7. Перевод в другую систему координат

Обратимся снова к рис. 11.1. Мы уже рассмотрели, каким образом преобразовывать координаты точек из модельных СК в мировую, теперь нам надо решать совсем другую

задачу. Точку  $P_w = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$  в мировой системе координат (МСК) надо представить в системе координат камеры (КСК)  $P_c = \begin{pmatrix} u \\ v \\ w \end{pmatrix}$ . Поскольку МСК и КСК могут иметь различную

ориентацию осей и разные начала координат, то данная задача распадается на следующие две:

1. Пусть МСК и КСК имеют одинаково ориентированные оси, а их точки начала различаются.
2. МСК и КСК имеют общее начало координат, но оси ориентированы по-разному.

### 11.7.1. Перенос начала координат

Пусть в мировой системе координат определены орты:  $\mathbf{x}, \mathbf{y}, \mathbf{z}$ . СК камеры имеет те же орты,

но начало находится в точке  $O = \begin{pmatrix} o_x \\ o_y \\ o_z \end{pmatrix}$  в мировой СК, тогда

$P_c = \begin{pmatrix} u \\ v \\ w \end{pmatrix} = O + u\mathbf{x} + v\mathbf{y} + w\mathbf{z} = x\mathbf{x} + y\mathbf{y} + z\mathbf{z}$ . Отсюда  $\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} x - o_x \\ y - o_y \\ z - o_z \end{pmatrix}$  или в однородных

координатах:

$$\begin{pmatrix} u \\ v \\ w \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & -o_x \\ 0 & 1 & 0 & -o_y \\ 0 & 0 & 1 & -o_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}. \quad (11.1)$$

Отсюда следует, что при переносе начала координат в точку  $O$  необходимо умножить на матрицу сдвига  $T(-o_x, -o_y, -o_z)$ , что не следует путать со сдвигом точки на вектор  $O$ , что обеспечивается матрицей  $T(o_x, o_y, o_z)$ .

### 11.7.2. Поворот системы координат

Мировая СК и СК камеры имеют общее начало, но разные орты у соответствующих осей:

$\mathbf{x}, \mathbf{y}, \mathbf{z}$  – МСК,  $\mathbf{u} = \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix}$ ,  $\mathbf{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$ ,  $\mathbf{w} = \begin{pmatrix} w_x \\ w_y \\ w_z \end{pmatrix}$  – КСК. Для точки  $P$  верно следующее равенство:

$x\mathbf{x} + y\mathbf{y} + z\mathbf{z} = u\mathbf{u} + v\mathbf{v} + w\mathbf{w}$ , или более подробно:

$$x = uu_x + vv_x + ww_x$$

$$y = uu_y + vv_y + ww_y.$$

$$z = uu_z + vv_z + ww_z$$

Приведем это выражение к матричному виду:  $\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix}$ . Матрица является

ортогональной, поэтому мы можем записать следующее равенство:  $\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ ,

или в однородных координатах:

$$\begin{pmatrix} u \\ v \\ w \\ 1 \end{pmatrix} = \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}. \quad (11.2)$$

Верхняя подматрица 3x3 – это ортогональная матрица вращения.

### 11.7.3. Преобразование мировых координат в СК камеры

Сразу отметим важный момент, что до сих пор мы рассматривали и МСК, и КСК как

правосторонние системы. Полное преобразование координат точки  $P_w = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$  из мировой СК

в СК камеры  $P_c = \begin{pmatrix} u \\ v \\ w \end{pmatrix}$  в однородных координатах выражается следующей формулой:

$$\begin{pmatrix} u \\ v \\ w \\ 1 \end{pmatrix} = \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -o_x \\ 0 & 1 & 0 & -o_y \\ 0 & 0 & 1 & -o_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}, \text{ или окончательно:}$$

$$\begin{pmatrix} u \\ v \\ w \\ 1 \end{pmatrix} = \begin{pmatrix} u_x & u_y & u_z & -u_x o_x \\ v_x & v_y & v_z & -u_y o_y \\ w_x & w_y & w_z & -u_z o_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}. \quad (11.3)$$

## 11.8. Видовое преобразование

Под видовым преобразованием понимается преобразование координат из мировой системы координат в систему координат результирующего изображения – порта вывода. Порт вывода (viewport) – это прямоугольная область на экране или клиентской области окна. Реально видовое преобразование раскладывается на несколько более простых, одно из которых – это преобразование камеры. Поскольку перевод из мировых координат в координаты СК камеры ужо рассмотрено в п.11.7.3, то перейдем к преобразованию камеры – перевод координат из СК камеры в координаты *картинной плоскости*, которая, как правило, перпендикулярна оси  $Z$ , рис. 11.8, при этом:

- При параллельном проецировании – это плоскость  $z = 0$ .
- При перспективном проецировании – это плоскость  $z = d > 0$ .

### 11.8.1. Параллельное проецирование

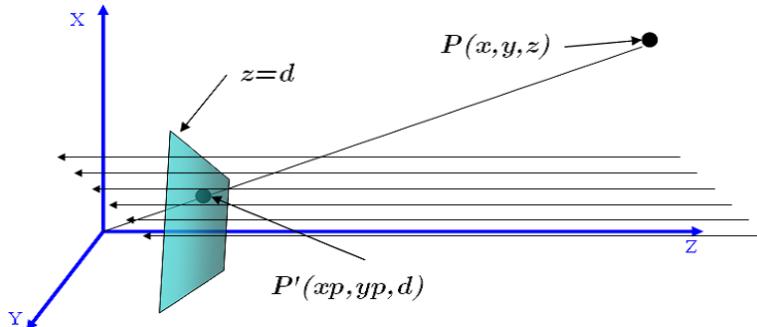


Рис. 11.8. Картичная плоскость и параллельное проецирование

Мы не рассматриваем все случаи, а только случай проецирования параллельно оси  $Z$  камеры. Точка  $P'(x_p, y_p, d)$  – это портрет точки  $P(x, y, z)$ . При этом  $x_p = x, y_p = y$ . В однородных координатах преобразование параллельного проецирования записывается

следующим образом:  $M_{ort} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ .

## 11.8.2. Перспективное проецирование

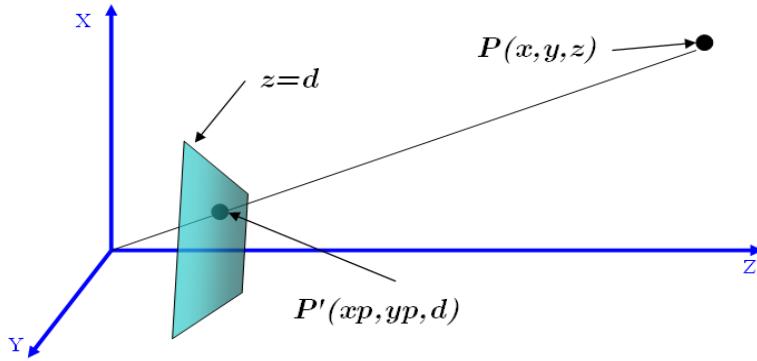


Рис. 11.9. Картичная плоскость и перспективное проецирование

Рассмотрим схему на рис. 11.9. Здесь считается, что координаты объектов заданы в системе координат камеры. Сама камера находится в начале координат, картинная плоскость  $z=d > 0$ . Направление вида (view direction) – ось  $Z$ . Точка картинной плоскости  $P'(x_p, y_p, d)$  является портретом точки сцены  $P(x, y, z)$ . Исходя из подобия треугольников, можно записать:

$$\frac{x_p}{d} = \frac{x}{z}; \frac{y_p}{d} = \frac{y}{z}; x_p = \frac{x}{z/d}; y_p = \frac{y}{z/d}; z \neq 0.$$

Покажем, что применяя матрицу проецирования на плоскость  $z=d$  в однородных

координатах:  $M_{psp}(d) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{pmatrix}$ , получим требуемый результат

$M_{psp}(d) \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ \frac{z}{d} \end{pmatrix} = \begin{pmatrix} X_p \\ Y_p \\ Z_p \\ W_p \end{pmatrix}$ . Вот! Именно здесь нам понадобилась однородная координата  $W_p$

$W_p = \frac{z}{d}$  отличная от 1. Для получения координат точки в  $R^3$  необходимо разделить на  $W_p$

остальные координаты однородного представления:  $\begin{pmatrix} X_p/W_p \\ Y_p/W_p \\ Z_p/W_p \\ W_p/W_p \end{pmatrix} = \begin{pmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ d \\ 1 \end{pmatrix} = \begin{pmatrix} x_p \\ y_p \\ d \\ 1 \end{pmatrix} = P'$ .

Покажем, что и в данном контексте можно опираться на однородные преобразования. Для этого рассмотрим другую постановку: камера находится в точке  $(0, 0, -d)$ , картинная плоскость имеет уравнение  $z=0$ , получить проецирующее преобразование  $M'_{psp}$ . Путем

несложных выводов получим, что  $M'_{psp}(d) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{d} & 1 \end{pmatrix}$ . Аналогичный результат мы

могли бы получить, если бы формально применили следующие действия: а) сдвинуть камеру в начало координат; б) спроектировать на плоскость  $z = d$ ; в) сдвинуть камеру обратно. Или в формульной записи:  $M'_{psp} = T(0, 0, -d)M_{psp}T(0, 0, d)$ .

### 11.8.3. Преобразование видимого объема камеры к полукубу

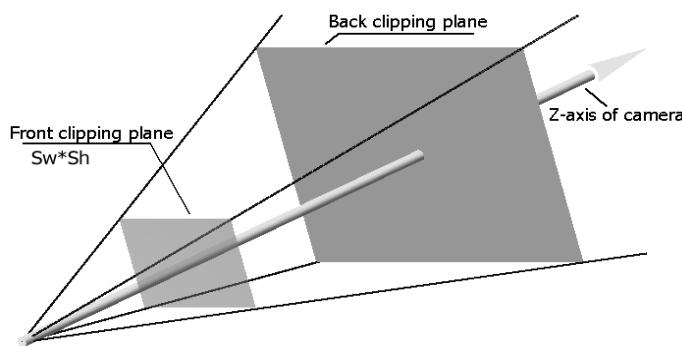


Рис. 11.10. Строение камеры

В большинстве современных программных продуктах, применяющихся для визуализации данных и 3D сцен, принята модель камеры, показанная на рис. 11.10. Рассмотрим ее поподробнее:

- Верхушка камеры, где располагается центр проекции, является вершиной неограниченной прямоугольной пирамиды, которая называется *пирамидой видимости* (pyramid of view – POV).
- *Направление вида* (view direction) или ось зрения – ось  $Z$  СК камеры.
- *Передняя/ближняя клиппирующая плоскость* (front/near clipping plane). Плоскость  $z = z_f > 0$ . Пирамида видимости вырезает на ней прямоугольник с размерами  $S_w \times S_h$ . Как правило, ось зрения проходит через центр этого прямоугольника. Этот прямоугольник в каком-то смысле является образом в СК камеры для порта вывода, т.е. при определении преобразования из мировых координат в экранные координаты надо считать, что порт вывода на экране – это прямоугольник  $S_w \times S_h$ .
- *Задняя/ дальняя клиппирующая плоскость* (back/far clipping plane). Плоскость  $z = z_b > z_f$ . Пирамида видимости вырезает на ней прямоугольник.
- *Видимый объем* (viewing frustum) – усеченная пирамида – часть пирамиды видимости, заключенная между передней и задней клиппирующими плоскостями. На порт вывода будут проецироваться только те объекты или части объектов сцены, которые находятся внутри видимого объема. Более того, остальные части объектов сцены не учитываются никак, даже при определении видимости. Например, между наблюдателем и передней плоскостью может находиться или нет какой-либо непрозрачный объект. В обоих случаях будут получены идентичные изображения.

Под *полукубом* здесь понимается следующий объем в пространстве –  $[-1, 1] \times [-1, 1] \times [0, 1]$ .

Первое, что мы можем сделать для определения изображения сцены, – это применить определенную выше матрицу проецирования  $M_{proj}(z_f)$ , т.е. передняя плоскость – это картинная плоскость. Нетрудно увидеть, что с учетом размеров передней клипирующей плоскости, все точки видимого объема и не только его перейдут в прямоугольник  $\left[-\frac{S_w}{2}, \frac{S_w}{2}\right] \times \left[-\frac{S_h}{2}, \frac{S_h}{2}\right] \times [z_f, z_b]$ . Этой информации уже достаточно для рисования, но тогда задача отсечения по видимому объему должна решаться до проецирования, что достаточно трудно для произвольной отсекающей пирамиды – ведь это отсечение по шести плоскостям в пространстве. Еще одно неудобство – это потеря глубины или дальности от камеры, что также заставляет решать задачу удаления невидимых линий и поверхностей до выполнения проецирования. С учетом этих соображений была предложена следующая матрица проецирования:

$$M_{proj}(S_w, S_h, z_f, z_b) = \begin{pmatrix} \frac{2}{S_w} z_f & 0 & 0 & 0 \\ 0 & \frac{2}{S_h} z_f & 0 & 0 \\ 0 & 0 & \frac{z_f}{z_b - z_f} & \frac{-z_f z_b}{z_b - z_f} \\ 0 & 0 & 1 & 0 \end{pmatrix}. \quad (11.4)$$

*Упражнение.* Показать, что видимый объем при помощи матрицы (11.4) переводится в полукуб  $[-1, 1] \times [-1, 1] \times [0, 1]$ .

Применим это преобразование к точке в СК камеры:

$$M_{proj}(S_w, S_h, z_f, z_b) \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \frac{2z_f}{S_w} \\ y \frac{2z_f}{S_h} \\ \frac{z \cdot z_b - z_f \cdot z_b}{z_b - z_f} \\ z \end{pmatrix} = \begin{pmatrix} X_p \\ Y_p \\ Z_p \\ W_p \end{pmatrix}.$$

Поделив на однородную координату  $W_p = z$ , получаем:

$$\begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix} = \begin{pmatrix} x \frac{2z_f}{S_w} \frac{1}{z} \\ y \frac{2z_f}{S_h} \frac{1}{z} \\ \left( \frac{z \cdot z_b - z_f \cdot z_b}{z_b - z_f} \right) \frac{1}{z} \\ 1 \end{pmatrix}.$$

Аналогичная матрица применяется и в случае параллельного проецирования:

$$M_{ortho}(S_w, S_h, z_f, z_b) = \begin{pmatrix} \frac{2}{S_w} & 0 & 0 & 0 \\ 0 & \frac{2}{S_h} & 0 & 0 \\ 0 & 0 & \frac{1}{z_b - z_f} & \frac{-z_f}{z_b - z_f} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (11.5)$$

$$\text{и } M_{ortho}(S_w, S_h, z_f, z_b) \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} X_p \\ Y_p \\ Z_p \\ W_p \end{pmatrix} = \begin{pmatrix} x \frac{2}{S_w} \\ y \frac{2}{S_h} \\ \frac{z}{z_b - z_f} - \frac{z_f}{z_b - z_f} \\ 1 \end{pmatrix}.$$

Таким образом, независимо от способа проецирования для любой точки сцены, принадлежащей видимому объему, на определенном этапе мы получим точку из полукуба.

#### 11.8.4. Определение камеры, орты

Теперь перейдем к определению камеры и построению ее системы координат. Опишем один из самых общепринятых способов задания системы координат камеры, опираясь на рис. 11.11.

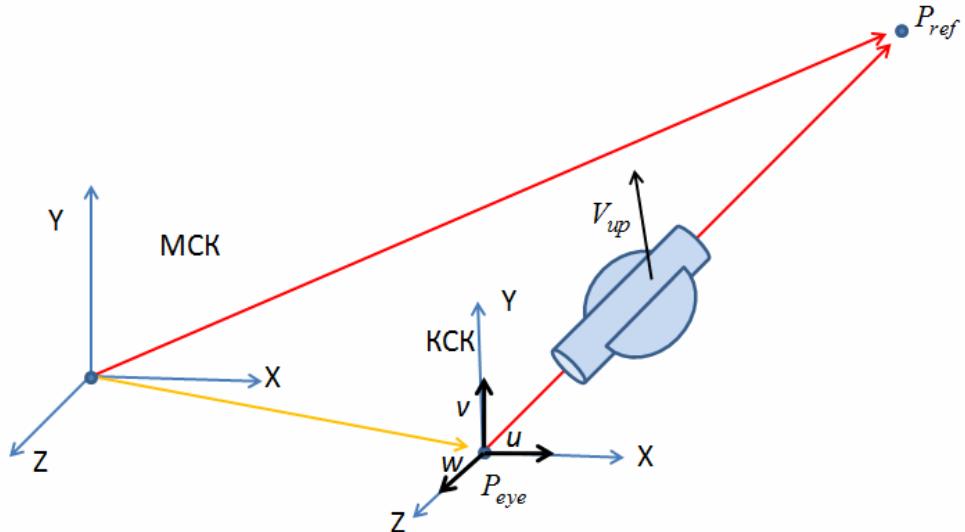


Рис. 11.11. Параметры задания камеры

На рисунке показаны две системы координат: мировая СК и видовая или СК камеры.  $P_{eye}$  – это точка наблюдателя, верхушка пирамиды видимости.  $P_{ref}$  – точка наблюдения, определяющая ось зрения. Для того чтобы определить вертикальную ось камеры  $Y$  задается "вектор вверх"  $V_{up}$ , который должен лежать в предполагаемой плоскости  $YZ$  СК камеры. Задает крен камеры ("самолетика"). Следующие выкладки показывают, как получаются орты СК камеры:

$$\mathbf{w} = \frac{P_{eye} - P_{ref}}{|P_{eye} - P_{ref}|}; \quad rr = V_{up} \times \mathbf{w}; \quad \mathbf{u} = \frac{rr}{|rr|}; \quad \mathbf{v} = \mathbf{w} \times \mathbf{u}.$$

Теперь надо только подставить  $O = P_{eye}$  в (11.3).

### 11.8.5. Корректировка преобразования проецирования

Снова обратимся к рис. 11.11., сравним с рис. 11.9 и 11.10. Ось  $Z$  камеры направлена по-разному – в противоположных направлениях. В документации по Direct3D такой ситуации не возникает, поскольку, в основном, там применяются левосторонние системы координат и для описания объектов, и в моделирующих преобразованиях, и в преобразовании камеры. В OpenGL моделирование ведется в правосторонней системе координат, а вот камера задается как бы безотносительно к правой/левой системе. По сравнению с рис. 11.9 в обеих упомянутых библиотеках придерживаются расположения осей, как на рис. 11.11, т.е. ось  $X$  направлена слева направо.

## 11.9. Алгоритм Z-буфера

В известной работе Сазерленда [16], посвященной 10 алгоритмам удаления невидимых линий и поверхностей, было отмечено, что эта задача не что иное, как сортировка по расстоянию от наблюдателя. Поэтому при визуализации пространственных сцен на одном из начальных этапов примитивы сортировались по глубине (расстоянию от камеры) или организовывались в определенные геометрические структуры, которые позволяли определять глубину на этапе изображения. После того, как память стала относительно недорогой, появилась возможность реализовать *алгоритм Z-буфера*.

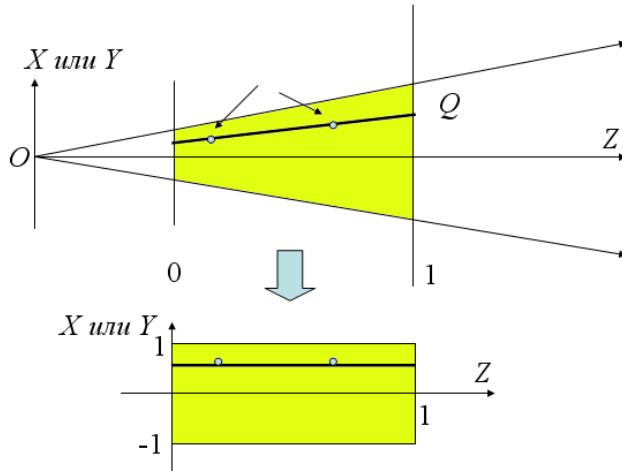


Рис. 11.12. Преобразование видимого объема в полукуб

Независимо от способа проецирования при помощи преобразований (11.4) и (11.5) мы получаем полукуб. Если при параллельном проецировании понятно, что части объектов, невидимые в пространстве сцены, будут невидимы и после перевода в полукуб – тут только масштабирование и сдвиг. С помощью рис. 11.12 мы покажем, что и в случае перспективного проецирования части объектов, невидимые в пространстве сцены, будут невидимы и после перевода в полукуб. При этом в полукубе будет применяться параллельное проецирование. Преобразование переводит точки видимого объема в точки полукуба. На луче  $\overrightarrow{OQ}$  выберем две точки, одна из них закрывает другую от камеры. Очевидно, что образ исходного луча – это луч параллельный оси  $Z$  в полукубе, и видимая точка луча имеет меньшее значение координаты  $Z$ . Таким образом, перспективное проецирование сведено к параллельному проецированию.

Второе замечательное качество полукуба – это простота отсечения частей объектов сцены, выходящих за пределы видимого объема. Теперь перейдем к самому алгоритму.

Порт вывода – это растр (или буфер кадра)  $R[N, M]$ , куда мы записываем значения цветов. Введем *буфер глубины* или *Z-буфер* – еще один массив (вещественный) такого же размера –  $Z_{buf}[N, M]$ , который будет хранить z-координату точки полукуба, чей цвет записан в растре.

*Алгоритм.*

*Начало.* Заносим цвет фона в буфер кадра  $R[N, M]$ . Заносим значение  $\infty$  во все элементы  $Z_{buf}[N, M]$ .

*Обработка очередной точки сцены.* Пусть "предположительно" изображаемая точка имеет цвет  $C$  и глубину  $z$  попадает в пиксель  $[i, j]$ . Тогда

```
if ( $Z_{buf}[i, j] < z$ ) {  $Z_{buf}[i, j] = z$  ;  $R[i, j] = C$  ; }
```

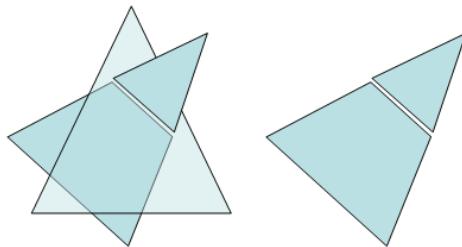


Рис. 11.13. Препроцессинг перед визуализацией

Следует отметить, что применение буфера глубины уменьшило затраты на предварительном этапе подготовки сцены к визуализации. Простейший пример показан на рис. 11.13. В сцене может возникнуть ситуация, когда один треугольник протыкает другой. В таком случае нельзя сказать, какой из них ближе – нельзя выстроить порядок. На фазе препроцессинга производится дробление примитивов, как показано на рис. 11.13 справа.

*Упражнение.* Показать, как с такой ситуацией справляется алгоритм Z-буфера.

## 11.10. Вращение вокруг произвольной оси

Выше отмечалось, что применение углов Эйлера для анимаций оказалось неприемлемым в основном из-за того, что представление вращения через углы Эйлера не однозначно. В связи с этим необходимо разработать другое описание поворота в пространстве вокруг произвольной оси.

**Вывод формулы вращения вокруг произвольной оси.**

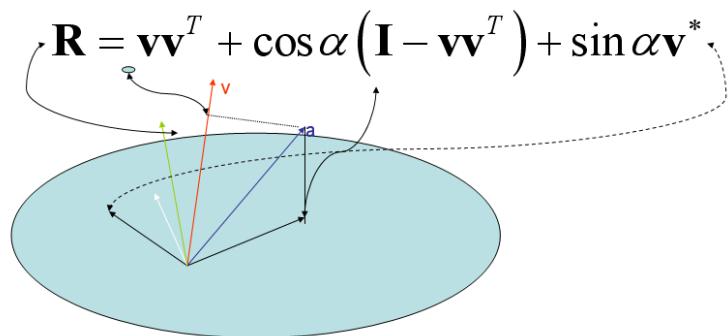


Рис. 11.14. Геометрическая интерпретация матрицы вращения

Для ненулевого вектора введем понятие *двойственной матрицы*:

$$\mathbf{v} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \Leftrightarrow \mathbf{v}^* = \begin{pmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix}. \text{ Легко проверить, что векторное произведение можно}$$

вычислить при помощи матричного преобразования:  $\mathbf{v}^* \mathbf{a} = \mathbf{v} \times \mathbf{a}$ . Будем использовать следующую геометрическую интерпретацию векторного произведения *оси вращения* (вектора  $\mathbf{v}$ ) на произвольный вектор  $\mathbf{a}$ :

1. Проецируем вектор  $\mathbf{a}$  на плоскость перпендикулярную  $\mathbf{v}$ .
2. Поворачиваем проекцию на 90 градусов.
3. Результат перпендикулярен и  $\mathbf{v}$ , и  $\mathbf{a}$ .

Покажем, что при помощи умножения любого вектора на матрицу  $\mathbf{R}(\mathbf{v}, \alpha) = \mathbf{v}\mathbf{v}^T + \cos \alpha (\mathbf{I} - \mathbf{v}\mathbf{v}^T) + \sin \alpha \mathbf{v}^*$  будет получаться вектор, повернутый на угол  $\alpha$  вокруг оси, определяемой вектором  $\mathbf{v}$ . Для этого обратимся к рис. 11.14:

- $\mathbf{v}$  – ось, единичный вектор.
- $\mathbf{v}\mathbf{v}^T$  – проекция на ось.
- $\mathbf{I} - \mathbf{v}\mathbf{v}^T$  – проекция на плоскость перпендикулярную к оси.
- $\mathbf{v}^*$  – проекция на плоскость перпендикулярную оси и поворот на 90 градусов.
- $\sin \alpha, \cos \alpha$  – задают угол поворота.

*Упражнение.* Самостоятельно расписать и проверить умножение матрицы  $\mathbf{R}$  на некоторый вектор  $\mathbf{a}$ .

## Литература

1. Sutherland I. SKETCHPAD, a man-machine graphical communication system // In. AFIPS Confer. Proc., SJCC. – 1963. – Vol. 23. – P. 329–346.
2. Bresenham J.E. Algorithm for computer control of digital plotter // IBM System Journal. – 1965. – Vol. 4, № 1. – P. 25–30.
3. Принс М.Д. Машиная графика и автоматизация проектирования. – М.: Советское радио, 1975.
4. Ньюмен У., Спрулл Р. Основы интерактивной машинной графики. – М.: Мир, 1976.
5. Status report of the Graphics Standards Planning Committee // Computer Graphics. – 1979. – Vol. 13, № 3.
6. ISO/DIS7942. ISO/TC 97. Information processing. – Graphical Kernel System (GKS). – Functional description. – Submitted on 1983-06-23.
7. ГОСТ 27817-88. Системы обработки информации. Машиная графика. Функциональное описание ядра графической системы. – М.: Госстандарт СССР, 1989.
8. Foley J.D., a.o. Computer graphics. Principles and practice – 2<sup>nd</sup> ed. in C. –Addison-Wesley, 1996.
9. Математический энциклопедический словарь под ред. Ю. В. Прохорова, — М.: Советская энциклопедия, 1988.
10. Wu, Xiaolin. An efficient antialiasing technique. // Computer Graphics. — 1991. — 25 (4). — P. 143–152.

11. Van Wyk C.J. Clipping to the boundary of a Circular-Arc polygon // Computer Vision, Graphics, and Image Processing. – Vol. 25 – 1984. – P. 383–392.
12. Препарата Шеймос статья о PTMO
13. Chernoff H. The Use of Faces to Represent Points in k-Dimensional Space Graphically. Journal of the American Statistical Association, 68(342):361-368, 1973.
14. Flury B., Riedwyl H. Graphical representation of multivariate data by means of asymmetrical faces, Journal of American Statistical Association, 76, 757-765, 1981.
15. Department of Computer Science Princeton University Adam Finkelstein, <http://www.cs.princeton.edu/courses/archive/spring05/cos426/>
16. Александр Амангельдыев. Требования к мониторам для издательских систем, [http://www.realcolor.ru/lib/kursiv\\_1-96/monitor1.shtml](http://www.realcolor.ru/lib/kursiv_1-96/monitor1.shtml)
17. Ivan E. Sutherland , Robert F. Sproull , Robert A. Schumacker, A Characterization of Ten Hidden-Surface Algorithms, ACM Computing Surveys (CSUR), v.6 n.1, p.1-55, March 1974.
18. <http://hyperfun.org/wiki/doku.php?id=frep:main>
19. Погорелов А.В. Дифференциальная геометрия 6е изд., Наука, 1974 180 С.