[1. Цели и задачи компьютерной графики](#_Toc294915678)

[2. Растровая и векторная визуализация изображений. Основные характеристики растровых изображений](#_Toc294915679)

[3. Аддитивная цветовая модель RGB. Кодирование цвета. Палитра](#_Toc294915680)

[4. Графические форматы. Работа с растровыми изображениями (форматы DDB, DIB, bmp-файлы)](#_Toc294915681)

[5. Понятие об инкрементных алгоритмах растеризации. Растровое представление отрезка. Вывод и реализация алгоритма](#_Toc294915682)

[6. Понятие об инкрементных алгоритмах растеризации. Построение окружности методом средней точки. Вывод и реализация алгоритма](#_Toc294915683)

[7. Стиль линии (перо). Алгоритм вывода линии](#_Toc294915684)

[8. Стиль заполнения, кисть, текстура](#_Toc294915685)

[9. Геометрические основы компьютерной графики. Системы координат (декартова, полярная, сферическая) и векторы. Скалярное и векторное произведение векторов](#_Toc294915686)

[10. Геометрические основы компьютерной графики. Уравнение прямой, проходящей через две точки (на плоскости). Параметрическое уравнение прямой. Уравнение отрезка. Задача о взаимном расположении двух отрезков](#_Toc294915687)

[11. Построение кривых. Интерполяционный полином Лагранжа. Достоинства и недостатки. Реализация](#_Toc294915688)

[12. Построение кривых. Интерполяционный кубический сплайн. Реализация](#_Toc294915689)

[12. Построение кривых. Геометрические сплайны. Кривая Безье. Геометрический алгоритм построения кривой Безье. Реализация](#_Toc294915690)

[13. Мировые и экранные координаты. Алгоритм пересчёта мировых 2D-координат в оконные. Вывод и реализация](#_Toc294915691)

[14. Физическая и логическая системы координат. Режимы отображения (без настройки параметров). Алгоритм преобразования координат в GDI и его связь с алгоритмом пересчёта мировых координат в оконные](#_Toc294915692)

[15. Режимы отображения и настройка их параметров (функции класса CDC MFC). Функция *SetMyMode*, назначение, параметры и реализация](#_Toc294915693)

[16. Аффинные преобразования на плоскости. Преобразование системы координат: смещение, растяжение-сжатие, поворот](#_Toc294915694)

[17. Аффинные преобразования на плоскости. Преобразование объектов: смещение, растяжение-сжатие, поворот](#_Toc294915695)

[18. Аффинные преобразования в пространстве. Преобразование системы координат: смещение, растяжение-сжатие, повороты вокруг осей координат](#_Toc294915696)

[19. Аффинные преобразования в пространстве. Преобразования объектов: смещение, растяжение-сжатие, повороты вокруг осей координат](#_Toc294915697)

[20. Основные типы проекций. Видовая система координат (вывод матрицы преобразования)](#_Toc294915698)

[21. Перспективные преобразования (вывод матрицы преобразования). Схема пересчёта координат при переходе от мировых 3D-координат к экранным](#_Toc294915699)

[22. Модели описания поверхностей](#_Toc294915700)

[23. Каркасная визуализация трёхмерных изображений. Принцип удаления невидимых граней для выпуклого многогранника. Реализация](#_Toc294915701)

[24. Изображение поверхности z = f(x,y). Метод сортировки граней по глубине (алгоритм художника). Реализация](#_Toc294915702)

[25. Закрашивание поверхностей. Модели отражения света](#_Toc294915703)

[26. Закрашивание поверхностей. Вычисление нормалей и углов отражения. Реализация](#_Toc294915704)

[27. Метод закрашивания Гуро](#_Toc294915705)

## 1. Цели и задачи компьютерной графики

Понятие "компьютерная графика" объединяет довольно широкий круг опе­раций по обработке графической информации с помощью компьютера. Причем наблюдается явная тенденция "компьютеризации" изображений, циркулирующих в обществе. Стали обыденностью термины "цифровое фото" и "видео". В виртуальных буднях грядущего компьютерной графике отводится огромная роль. Это свя­зано с тем, что, по мнению ученых, исследующих проблемы мозга, зритель­ная система в иерархии мозговых структур человека занимает особое место. С восприятием и обработкой визуальной информации непосредственно свя­зано примерно 20% мозга человека. Благодаря зрению мы получаем по раз­ным оценкам от 70 до 90% сведений об окружающем мире. Следовательно, образный мир компьютерной графики является одним из глубинных прояв­лений человеческой природы.

В компьютерной графике можно выделить несколько основных на­правлений:

* Визуализация научных данных

Большинство современных математических программных пакетов (Maple, MatLab) имеют средства для отображения гра­фиков, поверхностей и трехмерных тел, построенных на основе каких-либо расчетов. Кроме того, графическая информация может активно использоваться в самом процессе вычислений. Визуализация по­зволяет представить большой объем данных в удобной для анализа форме и широко используется при обработке результатов различных измерений и вычислений.

* Геометрическое проектирование и моделирование

Это направление компьютерной графики связано с решением задач начертательной гео­метрии – построением чертежей, эскизов, объемных изображений с по­мощью программных систем, получивших название CAD-системы (от английского Computer-Aided Design), например AutoCAD. Существует большое количество специализированных CAD-систем в ма­шиностроении, архитектуре и т. д.

* Распознавание образов

Способность распознавать абстрактные образы считают одним из важнейших факторов, определившим развитие мыслительных способностей человека, выделив его из животного мира. Задача распознавания и классификации графической информации явля­ется одной из ключевых и при создании искусственного интеллекта. Уже в наши дни компьютеры распознают образы повсеместно (системы идентификации футбольных хулиганов у входа на стадион; анализ аэро- и космических фотоснимков; системы сортировки, наведения и т. д.). Возможно, самый известный пример распознавания образов – сканиро­вание и перевод "фотографии'' текста в набор отдельных символов, фор­мирующих слова. Такую операцию позволяет выполнить программное обеспечение многих современных сканеров. Кроме того, сущест­вуют специализированные программы распознавания текста, например FineReader.

* Изобразительное искусство

К этому направлению можно отнести разно­образную графическую рекламу: от текстовых транспарантов и фирмен­ных знаков до компьютерных видеофильмов, обработку фотографий, соз­дание рисунков, мультипликацию и т. д. В качестве примера популярных программ из этой области компьютерной графики можно назвать Adobe Photoshop (обработка растровых изображений), CorelDRAW (создание векторной графики), 3DS Мах (трехмерное моделирование).

* Виртуальная реальность

Реальность, даже виртуальная, подразумевает воздействия на всю совокупность органов чувств человека, в первую оче­редь на его зрение. К компьютерной графике можно отнести задачи мо­делирования внешнего мира в различных приложениях: от компьютер­ных игр до тренажеров.

* Цифровое видео

Все более широкое распространение получают анимированные изображения, записанные в цифровом формате. Это прежде всего фильмы, передаваемые через компьютерные сети, а также видео­диски, цифровое, кабельное и спутниковое те­левидение.

Приведенная классификация сфер применения компьютерной графики яв­ляется во многом условной. Возможно, найдутся задачи, которые нельзя отнести ни к одному из обозначенных направлений.

## 2. Растровая и векторная визуализация изображений. Основные характеристики растровых изображений

Наиболее известны два способа визуализации: растровый и векторный. Первый способ ассоциируется с такими графическими устройствами, как дисплей, телевизор, принтер. Второй используется в векторных дисплеях, плоттерах.

Наиболее удобно, когда способ описания графического изображения соот­ветствует способу визуализации. Иначе нужна конвертация. Например, изо­бражение может храниться в растровом виде, а его необходимо вывести (ви­зуализировать) на векторном устройстве. Для этого нужна предварительная векторизация – преобразование из растрового в векторное описание. Или наоборот, описание изображения может быть в векторном виде, а нужно ви­зуализировать на растровом устройстве – необходима растеризация.

Растровая визуализация основывается на представлении изображения на эк­ране или бумаге в виде совокупности (массива) отдельных точек (пикселов). Вместе пикселы образуют растр.

Для обозначения массива пикселов часто используется термин bitmap (битовая карта). В bitmap каждому пикселу отводится определенное число битов (одинаковое для всех пикселов изображения). Это число назы­вается битовой глубиной пиксела или цветовой глубиной изображения, т. к. от количества битов, отводимых на один пиксел, зависит количество цветов изображения.

Векторная визуализация основывается на формировании изображения на эк­ране или бумаге рисованием линий (векторов) – прямых или кривых. Сово­купность типов линий (графических примитивов), которые используются как базовые для векторной визуализации, зависит от определенного устройства. Типичная последовательность действий при векторной визуализации для плоттера или векторного дисплея такова: переместить перо в начальную точ­ку (для дисплея – отклонить пучок электронов); опустить перо (увеличить яркость луча); переместить перо в конечную точку; поднять перо (уменьшить яркость луча).

Качество векторной визуализации для векторных устройств обуславливается точностью вывода и номенклатурой базовых графических примитивов – ли­ний, дуг, кругов, эллипсов и других.

Доминирующим сейчас является растровый способ визуализации. Это обу­словлено большей распространенностью растровых дисплеев и принтеров. Недостаток растровых устройств – дискретность изображения. Недостатки векторных устройств – проблемы при сплошном заполнении фигур, мень­шее количество цветов, меньшая скорость (в сравнении с растровыми уст­ройствами).

Растровые изображения и их основные характеристики

Растр – это матрица (массив) ячеек (пикселов). Каждый пиксел может иметь свой цвет. Совокупность пикселов различного цвета образует изображение. В за­висимости от расположения пикселов в пространстве различают квадратный, прямоугольный, гексагональный или иные типы растра. Для описания распо­ложения пикселов используют разнообразные системы координат. Общим для всех таких систем является то, что координаты пикселов образуют дис­кретный ряд значений (необязательно целые числа). Часто используется сис­тема целых координат – номеров пикселов с (0,0) в левом верхнем уголку. Такую систему мы будем использовать и в дальнейшем, ибо она удобна для рассмотрения алгоритмов графического вывода.

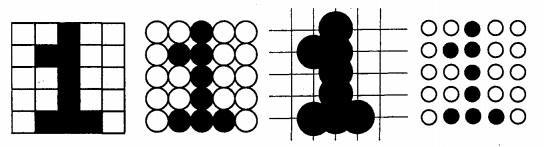
Геометрические характеристики растра

Разрешающая способность. Она характеризует расстояние между соседни­ми пикселами (см. рис. ниже). Разрешающую способность измеряют количеством пикселов на единицу длины. Наиболее популярной единицей измерения яв­ляется dpi (dots per inch) – количество пикселов в одном дюйме длины (2.54 см). Не следует отождествлять шаг с размерами пикселов – размер пикселов может быть равен шагу, а может быть как меньше, так и больше, чем шаг.



Размер растра обычно измеряется количеством пикселов по горизонтали и вертикали. Можно сказать, что для компьютерной графики зачастую наибо­лее удобен растр с одинаковым шагом для обеих осей, то есть dpiX = dpiY. Это удобно для многих алгоритмов вывода графических объектов.

Форма пикселов растра определяется особенностями устройства графиче­ского вывода (см. рис. ниже). Например, пикселы могут иметь форму прямоуголь­ника или квадрата, которые по размерам равны шагу растра (дисплей на жидких кристаллах); пикселы круглой формы, которые по размерам могут и не равняться шагу растра (принтеры).



Количество цветов (глубина цвета) – также одна из важнейших характери­стик растра. Количество цветов является важной характеристикой для любо­го изображения, а не только растрового. Согласно психофизиологическим исследованиям глаз человека способен различать 350000 цветов.

Классификация изображений:

* Двухцветные (бинарные) – 1 бит на пиксел. Среди двухцветных чаще всего встречаются черно-белые изображения.
* Полутоновые – градации серого или иного цвета. Например, 256 града­ций (1 байт на пиксел).
* Цветные изображения. От 2 бит на пиксел и выше. Глубина цвета 16 бит на пиксел (65 536 цветов) получила название High Color, 24 бит на пиксел (16,7 млн цветов) – True Color. В компьютерных графических системах используют и большую глубину цвета – 32, 48 и более бит на пиксел.

## 3. Аддитивная цветовая модель RGB. Кодирование цвета. Палитра

Для того чтобы компьютер имел возможность работать с цветными изобра­жениями, необходимо представлять цвета в виде чисел – кодировать цвет. Способ кодирования зависит от цветовой модели и формата числовых дан­ных в компьютере.

Чтобы оцифровать цвет, его необходимо измерить. Наука, которая изучает цвет и его измерения, называется колориметрией. Она описывает общие закономерности цветового восприятия света челове­ком. Одними из основных законов колориметрии являются законы смешивания цветов. Эти законы в наиболее полном виде были сформулированы в 1855 году немецким математиком Германом Грассманом:

* закон трехмерности – любой цвет может быть представлен ком­бинацией трех основных цветов;
* закон непрерывности – к любому цвету можно подобрать беско­нечно близкий;
* закон аддитивности – цвет смеси зависит только от цвета состав­ляющих.

Первый закон означает, что для любого заданного цвета (Color) можно записать такое цве­товое уравнение, выражающее линейную зависимость цветов:

Color = k1Color1 + k2Color2 + k3Color3,

где Color1, Color2, Color3 – некоторые базисные, линейно независимые цвета, коэффи­циенты k1 , k2 , k3 указывают количество соответствующего смешиваемого цвета. Линейная независимость цветов означает, что ни один из них не может быть выражен взвешенной суммой (линейной комбинацией) двух других.

Цветовая модель RGB

За основные три цвета приняты красный (Red), зеленый (Green), синий (Blue). В модели RGB любой цвет (Color) получается в результате сло­жения основных цветов.

Для модели RGB каждая из компонент может представляться числами, огра­ниченными некоторым диапазоном – например, дробными числами от 0 до 1 либо целыми числами от 0 до некоторого максимального значения. В настоящее время достаточно распространенным является формат True Color, в котором каждая компонента представлена в виде байта, что дает 256 градаций для каждой компоненты: R=0...255, G = 0...255, В = 0...255. Ко­личество цветов составляет 256\*256\*256 = 16.7 млн (224).

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

C = 00000000 bbbbbbbb gggggggg rrrrrrrr.

Цветовая модель RGB применяется для создания графических образов в устройствах, излучающих свет, - мониторах, телевизорах.

Цветовая модель CMYK. В полиграфических системах напечатанный на бумаге графический объект сам не излучает световых волн. Изобра­жение формируется на основе отраженной волны от окрашенных поверхностей. Окрашенные поверхности, на которые падает белый свет (т.е. сумма всех цветов), должны поглотить (т.е. вычесть) все составляющие цвета, кроме того, цвет которой мы видим. Цвет по­верхности можно получить красителями, которые поглощают, а не излучают. Например, если мы видим зеленое дерево, то это означа­ет, что из падающего белого цвета, т.е. суммы красного, зеленого, синего, поглощены красный и синий, а зеленый отражен. Цвета кра­сителей должны быть дополняющими:

* голубой (Cyan = В + G), дополняющий красного;
* пурпурный (Magenta = R + В), дополняющий зеленого;
* желтый (Yellow = R + G), дополняющий синего.

Но так как цветные красители по отражающим свойствам не одинаковы, то для повышения контрастности применяется еще чер­ный (Black). Модель CMYK названа по первым буквам слов Cyan, Magenta, Yellow и последней букве слова Black. Так как цвета вычи­таются, модель называется субстрактивной.

При работе с изображениями в системах компьютерной графики часто при­ходится искать компромисс между качеством изображения (требуется как можно больше цветов) и ресурсами, необходимыми для хранения и воспро­изведения изображения, исчисляемыми, например, объемом памяти (надо уменьшать количество бит на пиксел).

Кроме того, некоторое изображение само по себе может использовать огра­ниченное количество цветов. Например, для черчения может быть достаточ­но двух цветов, для человеческого лица важны оттенки розового, желтого, пурпурного, красного, зеленого; а для неба – оттенки голубого и серого. В этих случаях использование полноцветного кодирования цвета является избыточным.

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

## 4. Графические форматы. Работа с растровыми изображениями (форматы DDB, DIB, bmp-файлы)

Графическим форматом называют порядок (структуру), согласно которому данные, описывающие изображение, записаны в файле.

Типы форматов графических файлов определяются способом хранения и типом графических данных. Наиболее широко используются растровый, векторный и метафайловый форматы.

* Векторный формат наиболее удобен для хранения изображений, которые можно разложить на простые геометрические фигуры (например, чертежи или текст). Векторные файлы содержат математические описания элементов изображения. Наиболее распространенные векторные форматы: AutoCAD DXF и Microsoft SYLK.
* Растровый формат используется для хранения растровых данных. Файлы такого типа особенно хорошо подходят для хранения изображений реаль­ного мира, например оцифрованных фотографий. Растровые файлы содер­жат битовую карту изображения и се спецификацию. Наиболее распростра­ненные растровые форматы: BMP, TIFF, GIF, PCX, JPEG.
* Метафайловый формат позволяет хранить в одном файле и векторные, и растровые данные. Примером такого формата являются файлы CorelDRAW - CDR.

Кроме того, существуют файловые форматы для хранения мультипликации (видеоинформации), мультимедиа-форматы (одновременно хранят звуко­вую, видео- и графическую информацию), гипертекстовые (позволяют хра­нить не только текст, но и связи-переходы внутри него) и гипермедиа (ги­пертекст плюс графическая и видеоинформация) форматы, форматы трехмерных сцен, форматы шрифтов и т. д.

Формат файлов для хранения растровых изображений

К настоящему времени известно много форматов файлов для растровых изо­бражений. Здесь мы рассмотрим один из самых популярных форматов, кото­рый обязан своей распространенностью операционной системе Windows – формат BMP.

Общая структура ВМР – файла такова:



|  |  |
| --- | --- |
| BITIMAPFILEHEADER | 14 байт |
| BITMAPINFOHEADER | 40 байт |
| Палитра | Размер зависит от количества цветов |
| Битовый массив растрового изображения | Число байт определяется размерами растра и количеством бит на пиксел |

Заголовок файла BMP называется BITMAPFILEHEADER. В нем помещает­ся общее описание файла:

* код формата - символы "ВМ";
* общий размер файла в байтах;
* адрес битового массива в данном файле.

Далее в файле следует еще один заголовок – BITMAPINFOHEADER, в котором хранится описание размеров растра и цветового формата пикселов:

* размер заголовка;
* ширина растра в пикселах;
* высота растра в пикселах;
* битовая глубина пиксела (бит/пиксел);
* размер в байтах битового массива растра.

Затем в файле помещается палитра в виде записей RGBQUAD. Каждая запись содержит значения каждой из цветовых компонент модели RGB в виде числа 0…255. Количество записей RGBQUAD равно количеству используемых цветов.

Па­литра отсутствует, если число бит на пиксел равно 24. Также палитра не нужна и для некоторых цветовых форматов 16 и 32 бит на пиксел.

После палитры (если она есть) в файле BMP записывается растр в виде бито­вого (а точнее, байтового массива). В битовом массиве последовательно за­писываются байты строк растра. Количество байт в строке должно быть кратно четырем, поэтому если количество пикселов по горизонтали не соот­ветствует такому условию, то справа в каждую строку дописывается некото­рое число битов (выравнивание строк на границу двойного слова).

Форматы DDB и DIB

В операционной системе Windows используются два формата битовых изображений – аппаратно-зависимый DDB (device-dependent bitmap) и аппаратно-независимый DIB (device-independent bitmap).

Согласно определению, данному в документации к SDK, битовое изображение DDB есть набор бит в оперативной памяти, который может быть отображен на устройстве вывода(например, выведен на экран видеомонитора или распечатан на принтере). Внутренняя структура изображения DDB жестко привязана к аппаратным особенностям устройства вывода. Поэтому представление изображения DDB в оперативной памяти полностью зависит от устройства вывода.

Иногда битовые изображения называют растровыми изображениями, подчеркивая тот факт, что его можно рассматривать как совокупность строк растра (горизонтальных линий развертки).

Если бы в Windows можно было работать только с изображениями DDB, было бы необходимо иметь отдельные наборы изображений для каждого типа видеоконтроллера и каждого видеорежима, что, очевидно, крайне неудобно.

Аппаратно-независимое битовое изображение DIB содержит описание цвета пикселов изображения, которое не зависит от особенностей устройства отображения. Операционная система Windows после соответствующего преобразования может отобразить такое изображение на любом устройстве вывода. Несмотря на некоторое замедление процесса вывода по сравнению с выводом изображений DDB, универсальность изображений DIB делает их весьма привлекательными для хранения изображений.

Битовые изображения в формате DDB

Битовые изображения в формате DDB являются аппаратно-зависимыми. Поэтому структура изображения в оперативной памяти зависит от особенностей аппаратуры.

Как правило, изображения DDB либо загружаются из ресурсов приложения, либо создаются непосредственно в оперативной памяти. Для вывода изображений DDB на экран используются такие функции, как BitBlt и StretchBlt.

BOOL WINAPI BitBlt(

HDC hdcDest, // контекст для рисования

int nXDest, // x-координата верхнего левого угла

// области рисования

int nYDest, // y-координата верхнего левого угла

// области рисования

int nWidth, // ширина изображения

int nHeight, // высота изображения

HDC hdcSrc, // идентификатор исходного контекста

int nXSrc, // x-координата верхнего левого угла

// исходной области

int nYSrc, // y-координата верхнего левого угла

// исходной области

DWORD dwRop) // код растровой операции

Функция копирует битовое изображение из исходного контекста hdcSrc в контекст отображения hdcDest. Возвращаемое значение равно TRUE при успешном завершении или FALSE при ошибке.

В качестве кода растровой операции чаще всего используется константа SRCCOPY. При этом цвет пикселов копируемого изображения полностью замещает цвет соответствующих пикселов контекста отображения. В этом случае цвет кисти, выбранной в контекст отображения, не имеет значения, так как ни цвет кисти, ни цвет фона не влияют на цвет нарисованного изображения.

Для рисования битовых изображений можно использовать вместо функции BitBlt функцию StretchBlt , с помощью которой можно выполнить масштабирование (сжатие или растяжение) битовых изображений:

BOOL WINAPI StretchBlt(

HDC hdcDest, // контекст для рисования

int nXDest, // x-координата верхнего левого угла

// области рисования

int nYDest, // y-координата верхнего левого угла

// области рисования

int nWidthDest, // новая ширина изображения

int nHeightDest, // новая высота изображения

HDC hdcSrc, // идентификатор исходного контекста

int nXSrc, // x-координата верхнего левого угла

// исходной области

int nYSrc, // y-координата верхнего левого угла

// исходной области

int nWidthSrc, // ширина исходного изображения

int nHeightSrc, // высота исходного изображения

DWORD dwRop) // код растровой операции

Параметры этой функции аналогичны параметрам функции BitBlt, за исключением того, что ширина и высота исходного и полученного изображения должна определяться отдельно. Размеры исходного изображения (логические) задаются параметрами nWidthSrc и nHeightSrc, размеры нарисованного изображения задаются параметрами nWidthDest и nHeightDest.

Возвращаемое значение равно TRUE при успешном завершении или FALSE при ошибке.

Битовые изображения в формате DIB

Изображения DIB, в отличие от изображений DDB, являются аппаратно-независимыми, поэтому без дополнительного преобразования их нельзя отображать на экране с помощью функций BitBlt и StretchBlt. В операционной системе Windows битовые изображения хранятся в файлах с расширением имени bmp, при этом используется аппаратно-независимый формат DIB.

Файл, содержащий битовое изображение, начинается со структуры BITMAPFILEHEADER. Эта структура описывает тип файла и его размер, а также смещение области битов изображения.

typedef struct tagBITMAPFILEHEADER {

WORD bfType; // Признак BMP – файла, 42 4D (коды букв ВМ)

DWORD bfSize; // Размер файла

WORD bfReserved1; // Первое резервное поле - всегда ноль.

WORD bfReserved2; // Второе резервное поле - тоже ноль

DWORD bfOffBits; // Смещение от начала файла до первого

// байта графических данных

} BITMAPFILEHEADER, \*PBITMAPFILEHEADER;

Сразу после структуры BITMAPFILEHEADER в файле следует структура BITMAPINFO, которая содержит описание изображения и таблицу цветов.

typedef struct tagBITMAPINFO {

BITMAPINFOHEADER bmiHeader;

RGBQUAD bmiColors[1];

} BITMAPINFO, \*PBITMAPINFO;

Описание изображения (размеры изображения, метод компрессии, размер таблицы цветов и т. д.) находится в структуре BITMAPINFOHEADER.

typedef struct tagBITMAPINFOHEADER{

DWORD biSize; // Размер BITMAPINFOHEADER в байтах.

LONG biWidth; // Ширина картинки в пикселах

LONG biHeight; // Высота картинки в пикселах

WORD biPlanes; // Количество битовых плоскостей (=1, см. MSDN)

WORD biBitCount; // Количество бит на пиксел (определяет

// Mаксимальное число цветов в bitmap)

DWORD biCompression; // Cпособ сжатия (0 - данные не сжимаются).

DWORD biSizeImage; // Размер изображения в байтах (без заголовков)

LONG biXPelsPerMeter; // Число пикселов на метр по горизонтали

LONG biYPelsPerMeter; // Число пикселов на метр по вертикали

DWORD biClrUsed; // Количество элементов палитры, хранящихся

// после заголовка

DWORD biClrImportant; // Количество разных цветов, действительно

// используемых в рисунке.

} BITMAPINFOHEADER, \*PBITMAPINFOHEADER;

В некоторых случаях (не всегда) в файле может присутствовать таблица цветов (как массив структур RGBQUAD), присутствующих в изображении.

typedef struct tagRGBQUAD

{

BYTE rgbBlue;

BYTE rgbGreen;

BYTE rgbRed;

BYTE rgbReserved;

} RGBQUAD;

Биты изображения обычно располагаются сразу после таблицы цветов. Точное значение смещения битов изображения находится в структуре BITMAPFILEHEADER.

Загрузка данных из BMP файла

Чтобы изображение загрузить с диска в оперативную память и получить дескриптор изображения (типа HBITMAP), предназначены функции LoadBitmap() и LoadImage().

HBITMAP Bit = (HBITMAP)LoadImage(NULL, char\* Filename, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE | LR\_CREATEDIBSECTION)

Значение NULL первого параметра указывает, что картинка вводится из файла. При получении изображения из ресурса в первом параметре указывается дескриптор приложения. Во втором параметре задается имя файла или идентификатор ресурса. Третий параметр задает тип изображения и может принимать значения IMAGE\_BITMAP, IMAGE\_CURSOR, IMAGEICON. Четвертый и пятый параметры, которые указывают размеры изображения, при вводе битовой карты задаются нулями, так как эта информация имеется в заголовке BMP-файла. Шестой параметр содержит сочетание флагов. В частности, флаг LR\_LOADFROMFILE указывает, что изображение вводится из файла. При задании флага LR\_CREATEDIBSECTION создается аппаратно-независимое, а при его отсутствии – аппаратно-зависимое растровое изображение. Ниже приводится код функции для загрузки битовой карты, которая получает путь к файлу FileName и тип растра Туре. Нулевое значение будет означать, что используется тип DIB, а единичное значение будет применяться для типа DDB.

HBITMAP LoadPict(char \*FileName, int Type)

{

if(Type)

return LoadImage(NULL, FileName,

IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE);

else return Loadlmage(NULL, FileName, IMAGE\_BITMAP, 0, 0,

LR\_LOADFROMFILE|LR\_CREATEDIBSECTION);

}

Функция LoadImage() позволяет загружать из ресурса битовую карту, курсор или пиктограмму, но для этого более удобны простые специализированные функции LoadBitmap(), LoadCursor(), LoadIcon().

## 5. Понятие об инкрементных алгоритмах растеризации. Растровое представление отрезка. Вывод и реализация алгоритма

Алгоритмы компьютерной графики можно разделить на два уровня: нижний и верхний. Группа алгоритмов нижнего уровня предназначена для реализации графических примитивов (линий, окружностей, заполнений и т.п.). Эти алгоритмы или подобные им воспроизведены в графических библиотеках языков высокого уровня или реализованы аппаратно в графических процессорах рабочих станций.

Среди алгоритмов нижнего уровня можно выделить следующие группы:

* I группа – используют простые математические методы и отличаются простотой реализации. Как правило, такие алгоритмы не являются наилучшими по объему выполняемых вычислений или требуемым ресурсам памяти.
* II группа – используют более сложные математические методы по сравнению с алгоритмами I группы (но часто и эвристические) и отличаются большей эффективностью.
* III группа – алгоритмы, которые могут быть без больших затруднений реализованы аппаратно (допускающие распараллеливание, рекурсивные, реализуемые в простейших командах). В эту группу могут попасть и алгоритмы, представленные в первых двух группах.

К алгоритмам верхнего уровня относятся в первую очередь алгоритмы удаления невидимых линий и поверхностей. Задача удаления невидимых линий и поверхностей продолжает оставаться центральной в машинной графике. От эффективности алгоритмов, позволяющих решить эту задачу, зависят качество и скорость построения трехмерного изображения.

К задаче удаления невидимых линий и поверхностей примыкает задача построения (закрашивания) полутоновых (реалистических) изображений, т.е. учета явлений, связанных с количеством и характером источников света, учета свойств поверхности тела (прозрачность, преломление, отражение света).

Однако при этом не следует забывать, что вывод объектов в алгоритмах верхнего уровня обеспечивается примитивами, реализующими алгоритмы нижнего уровня, поэтому нельзя игнорировать проблему выбора и разработки эффективных алгоритмов нижнего уровня.

Для разных областей применения компьютерной графики на первый план могут выдвигаться разные свойства алгоритмов. Для научной графики большое значение имеет универсальность алгоритма, быстродействие может отходить на второй план. Для систем моделирования, воспроизводящих движущиеся объекты, быстродействие становится главным критерием, поскольку требуется генерировать изображение практически в реальном масштабе времени.

Особенности растровой графики связаны с тем, что обычные изображения, с которыми сталкивается человек в своей деятельности (чертежи, графики, карты, художественные картины и т.п.), реализованы на плоскости, состоящей из бесконечного набора точек. Экран же растрового дисплея представляется матрицей дискретных элементов, имеющих конкретные физические размеры. При этом число их существенно ограничено. Поэтому нельзя провести точную линию из одной точки в другую, а можно выполнить только аппроксимацию этой линии с отображением ее на дискретной матрице (плоскости). Такую плоскость также называют целочисленной решеткой, растровой плоскостью или растром. Эта решетка представляется квадратной сеткой с шагом 1. Отображение любого объекта на целочисленную решетку называется разложением его в растр или просто растровым представлением.

Базовые растровые алгоритмы

Сформировать растровое изображение можно по-разному. Для того чтобы создать изображение на растровом дисплее, можно просто скопировать гото­вый растр в видеопамять. Этот растр может быть получен, например, с по­мощью сканера или цифрового фотоаппарата. А можно создавать изобра­жение объекта путем последовательного рисования отдельных простых эле­ментов.

Простые элементы, из которых складываются сложные объекты, называют графическими примитивами. Их можно встретить повсюду. Например, для построения изображения используется некоторый набор примитивов, ко­торые поддерживаются определенными графическими устройствами вывода. Графические примитивы также можно применять для описания пространст­венных объектов в базе данных компьютерной системы – модели объектов. Могут использоваться различные множества примитивов для модели и для алгоритмов отображения. Удобно, когда эти множества совпадают, тогда уп­рощается процесс отображения.

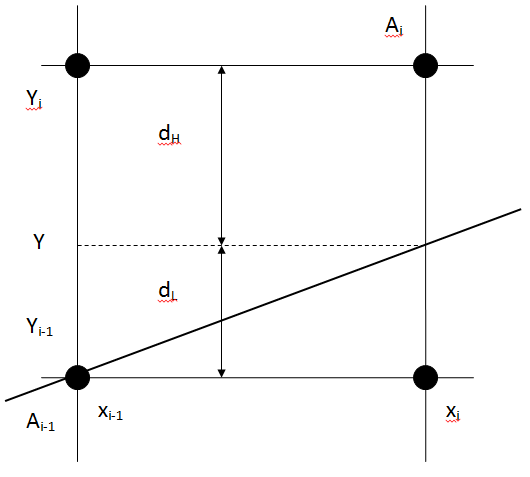
Простейшим и, вместе с тем, наиболее универсальным растровым графиче­ским примитивом является пиксел. Любое растровое изображение можно на­рисовать по пикселам, но это сложно и долго. Необходимы больше сложные элементы, для которых рисуются сразу несколько пикселов.

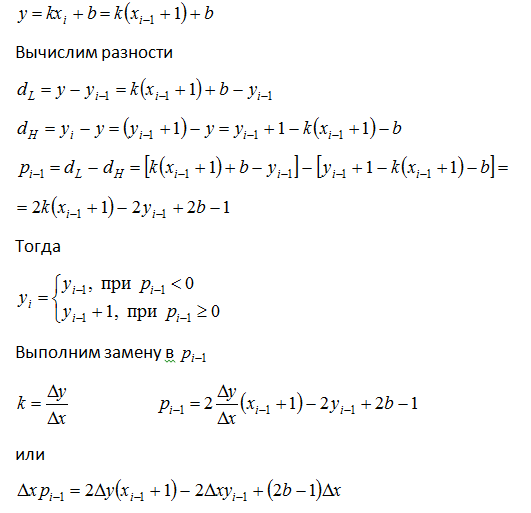
Инкрементные алгоритмы растеризации

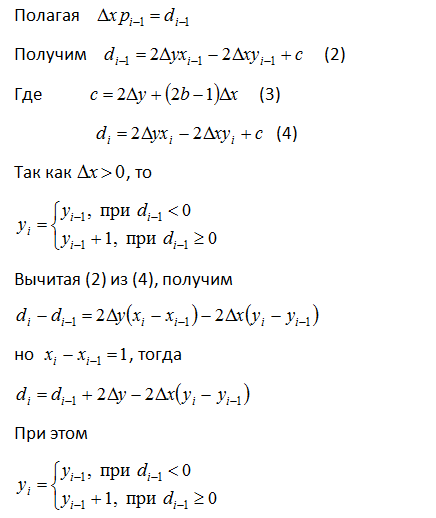
Брезенхэм предложил подход, позволяющий разрабатывать так назы­ваемые инкрементные алгоритмы растеризации. Основной целью для разра­ботки таких алгоритмов было построение циклов вычисления координат на основе только целочисленных операций сложения/вычитания без использо­вания умножения и деления. Инкрементные алгоритмы выполняются как последовательное вычисление координат соседних пикселов путем добавления приращений координат.

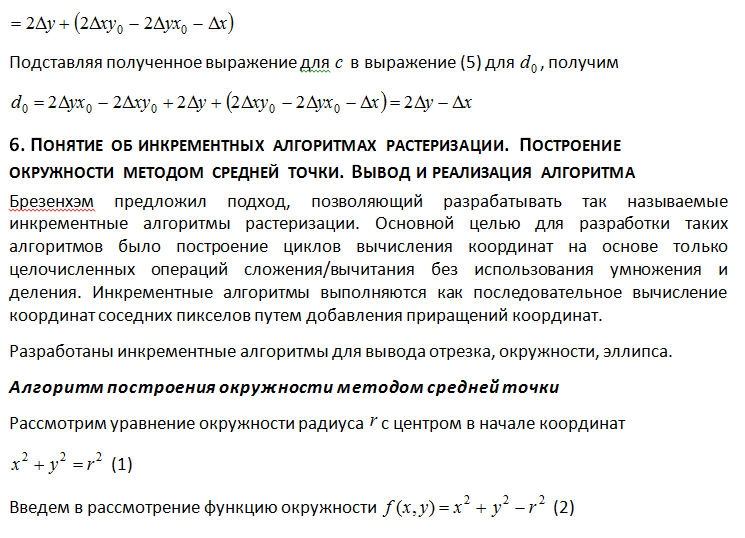
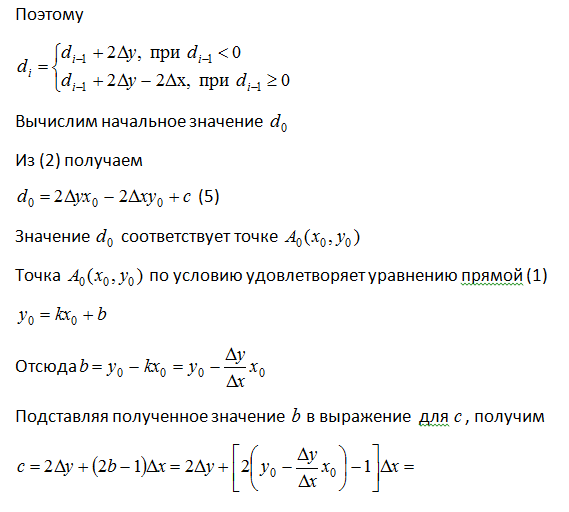
Разработаны инкрементные алгоритмы для вывода отрезка, окружности, эллипса.

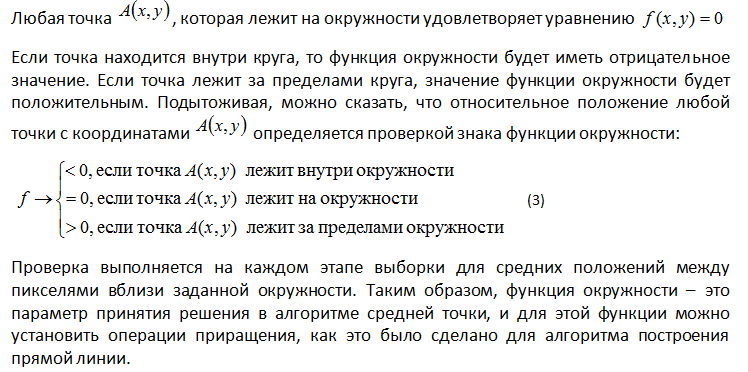
Алгоритм вывода прямой линии

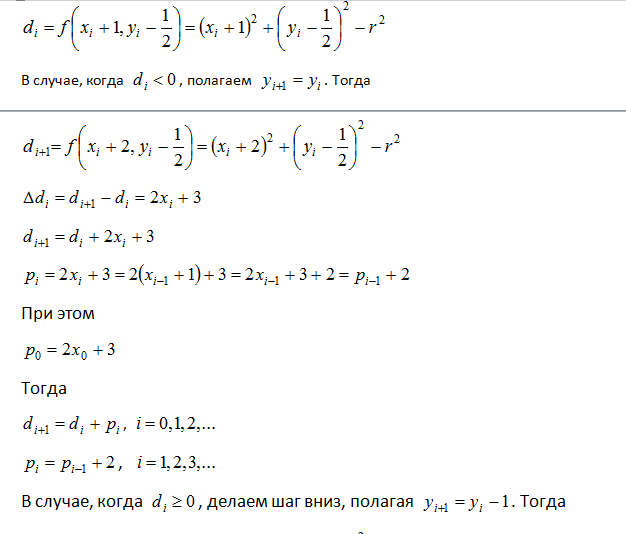
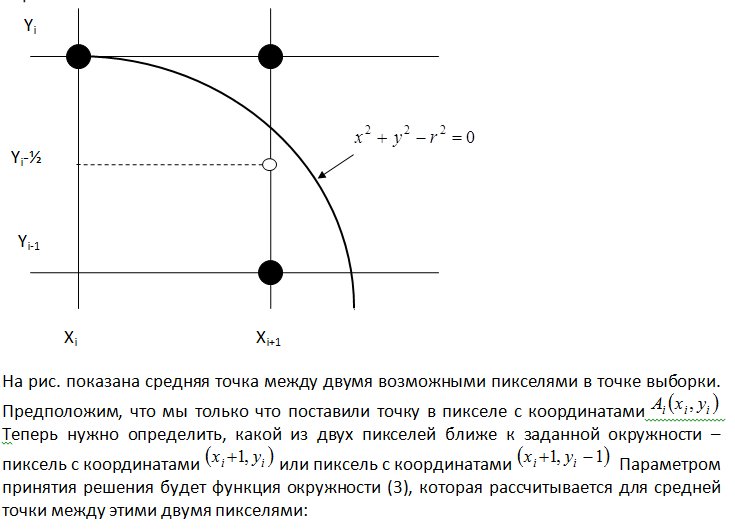


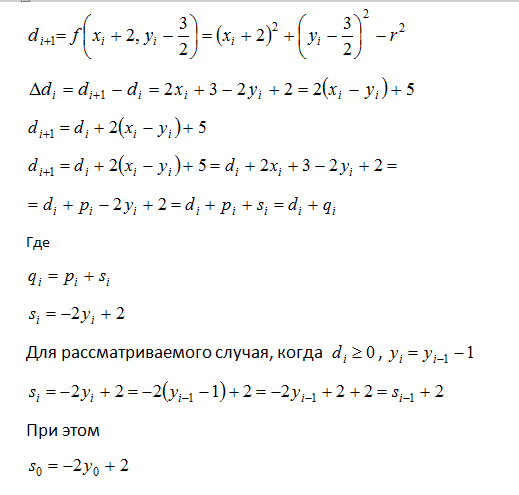


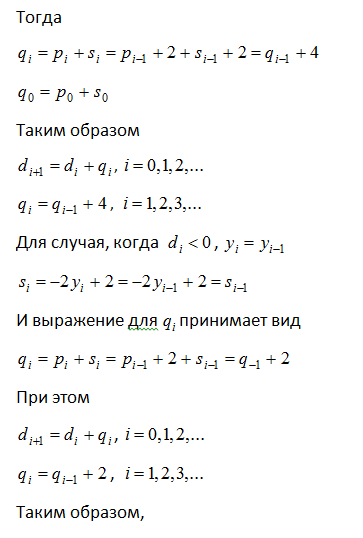


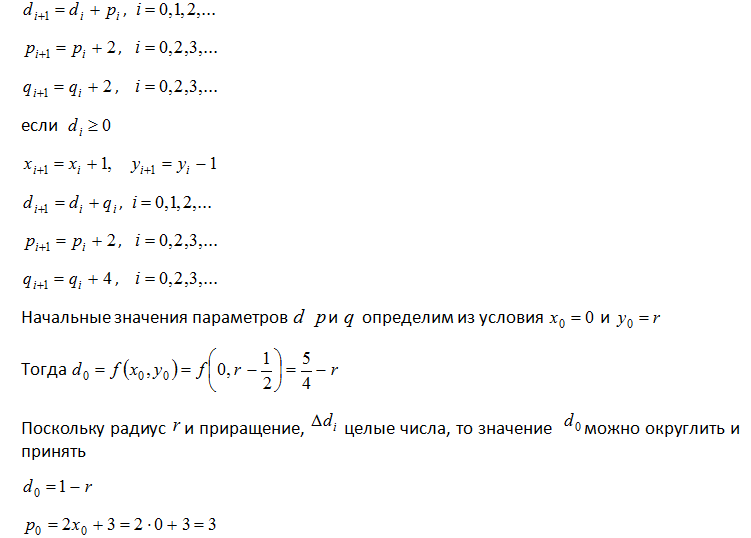
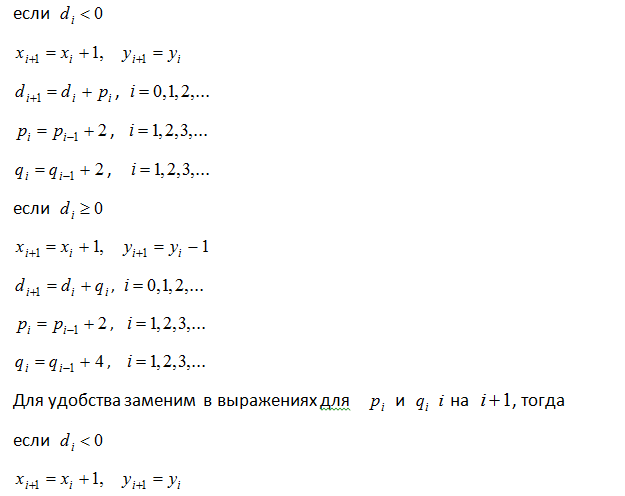


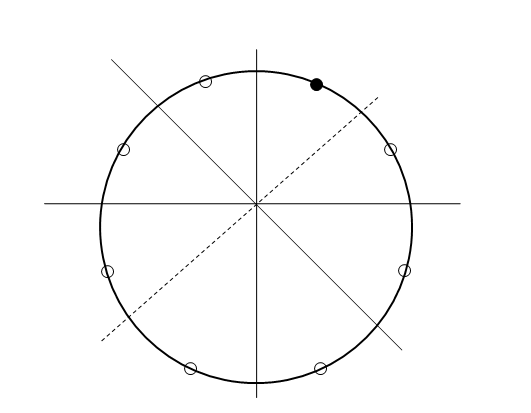
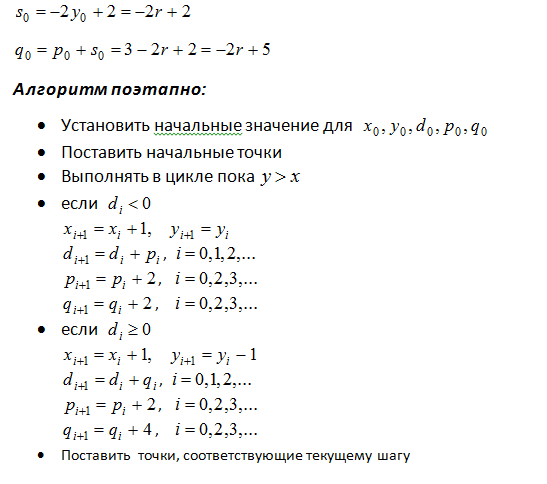




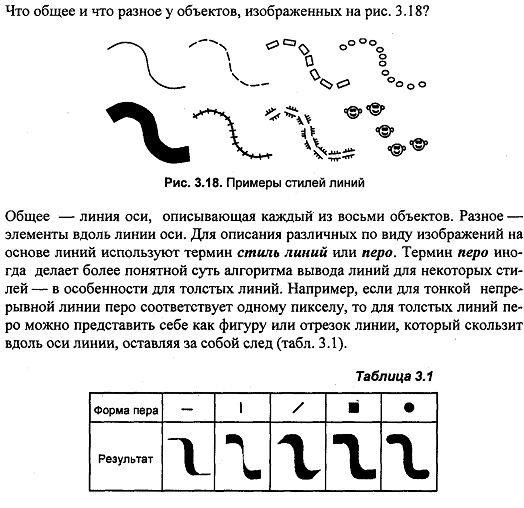


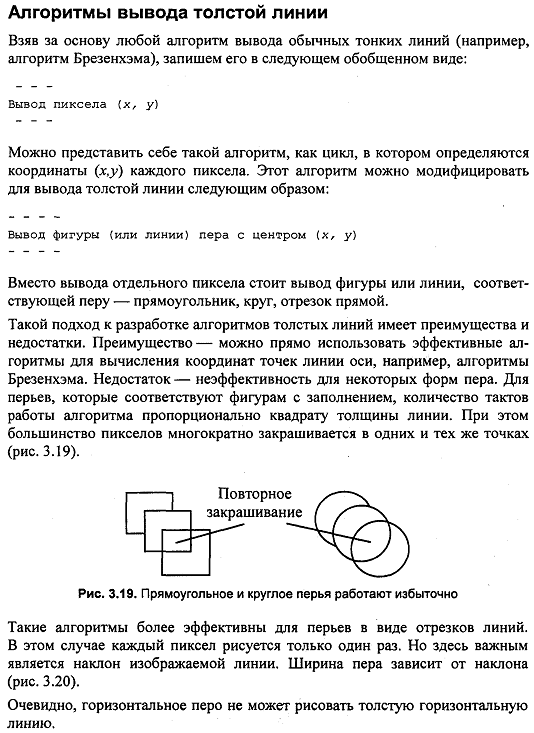


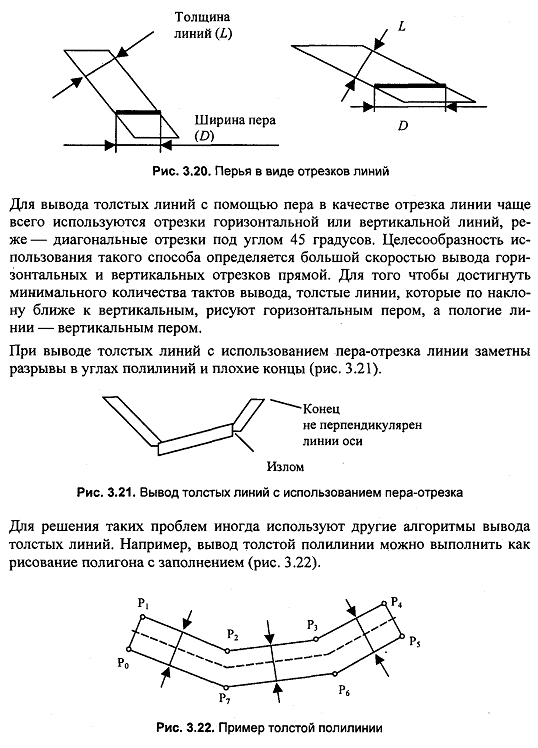




## 7. Стиль линии (перо). Алгоритм вывода линии







## 8. Стиль заполнения, кисть, текстура

Для обозначения стилей заполнения, отличных от сплошного стиля, используют такие понятия, как кисть и текстура. Текстура – это стиль заполнения, закрашивание, которое имитирует сложную рельефную объемную поверхность, выполненную из какого-то материала.

Для описания алгоритмов заполнения фигур с определенным стилем исполь­зуем следующий алгоритм рисования линий:

* вывод пиксела заполнения цвета C с координатами (х, у)

Например, в алгоритме вывода полигонов пикселы заполнения рисуются в теле цикла горизонталей, а все другие операции предназначены для подсчета координат (х, у) этих пикселов. Сплошное заполнение означает, что цвет (С) всех пикселов одинаков, то есть C = const. Преобразуем ал­горитм заполнения следующим образом:

* С = f (х, у)
* вывод пиксела заполнения (х, у) цветом C.

Функция f(x, y) будет определять стиль заполнения. Аргументами функции цвета являются координаты текущего пиксела заполнения. Однако в отдельных случаях эти аргументы не нужны. Например, если цвет C вычисляется как случайное значение в определённых границах: С = random(), то можно создать иллюзию шершавой матовой поверхности (рис. 3.23).



Другой стиль заполнения – штриховой. Для него функцию цвета также можно записать в аналитической форме:



где *S* – период, а *T* – толщина штрихов, *Cш* – цвет штрихов, *Cф* – цвет фона.

Зачастую при использовании кистей и текстур используется копирование не­больших растровых изображений. Такой алгоритм заполнения можно опи­сать вышеупомянутой общей схемой, если строку С = f (х, у) заменить двумя другими строками:

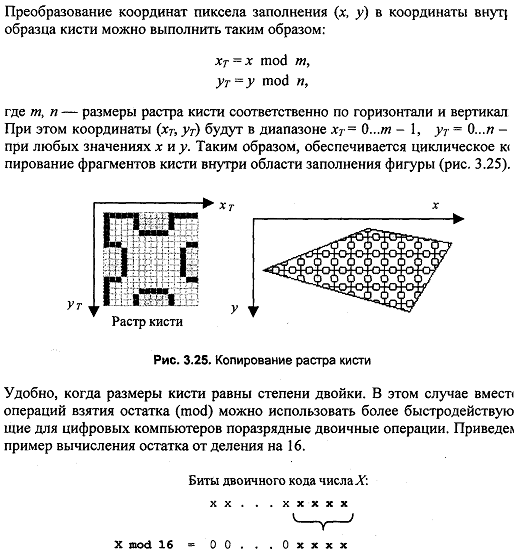
* координаты пиксела заполнения (х, у) преобразуем в растровые координаты образца кисти (хт / yт*)*
* по координатам (хт / yт*)* определяем цвет (С) пиксела в образце кисти
* вывод пиксела заполнения цвета С c координатами (х, у)

Преобразование координат пиксела заполнения (х, у) в координаты внутри образца кисти можно выполнить таким образом:

хт *= х* mod m,

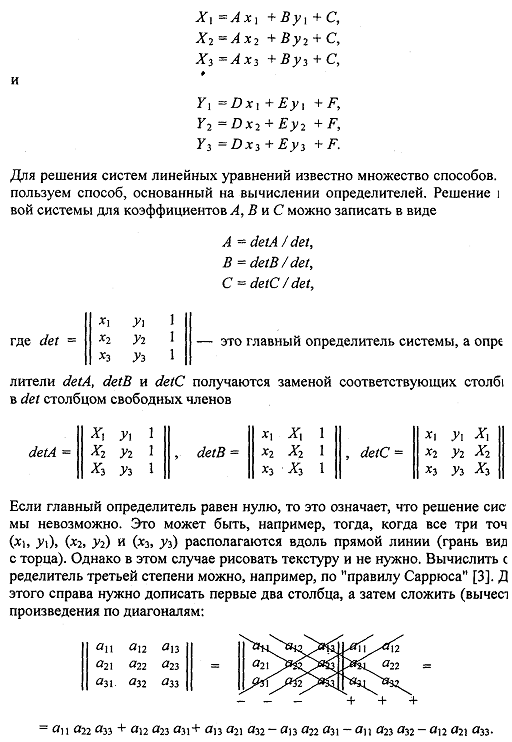
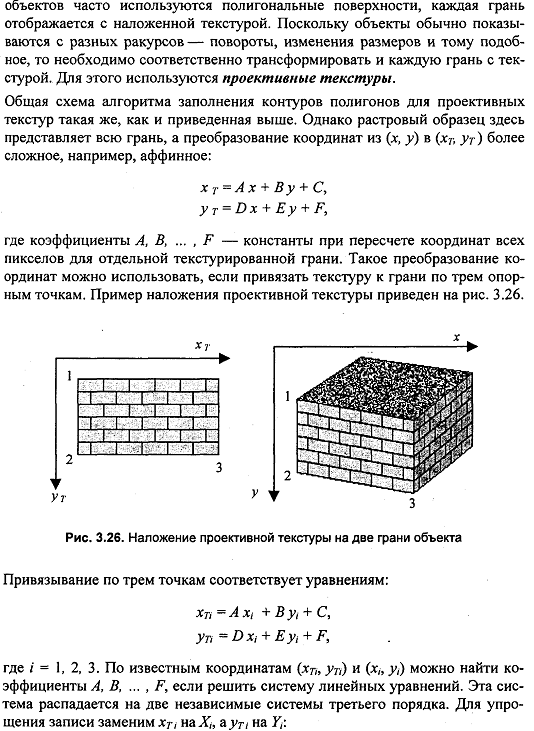
yт *= y* mod n*,*

где m, n – размеры растра кисти соответственно по горизонтали и вертикали. При этом координаты (хт, yт*)* будут в диапазоне хт *=* 0...m - 1, yт *=* 0...n –при любых значениях х и у.Таким образом, обеспечивается циклическое копирование фрагментов кисти внутри области заполнения фигуры.



Проективные текстуры

Поскольку объекты часто показываются с разных ракурсов – повороты, изменения размеров и т. п., то необходимо трансформировать соответственно и каждую грань с текстурой. Для этого используются проективные текстуры.

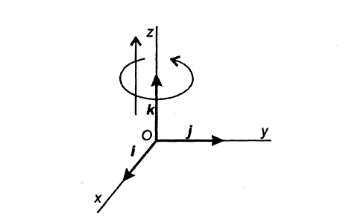


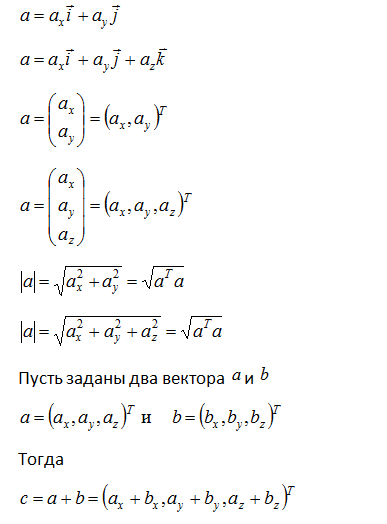
## 9. Геометрические основы компьютерной графики. Системы координат (декартова, полярная, сферическая) и векторы. Скалярное и векторное произведение векторов

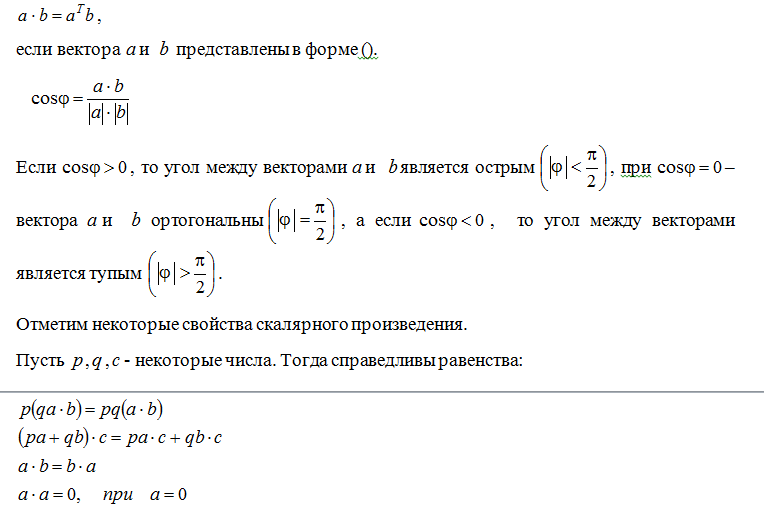
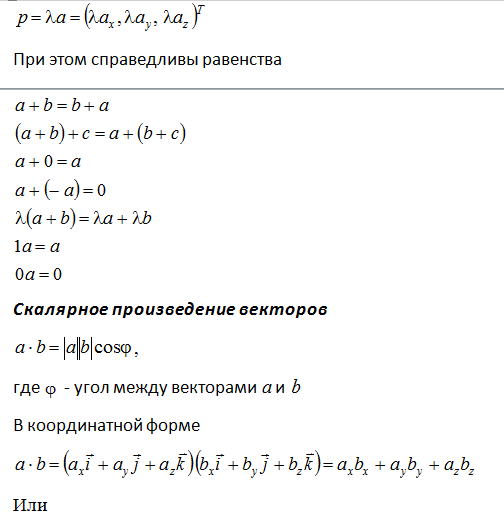
Системы координат и векторы

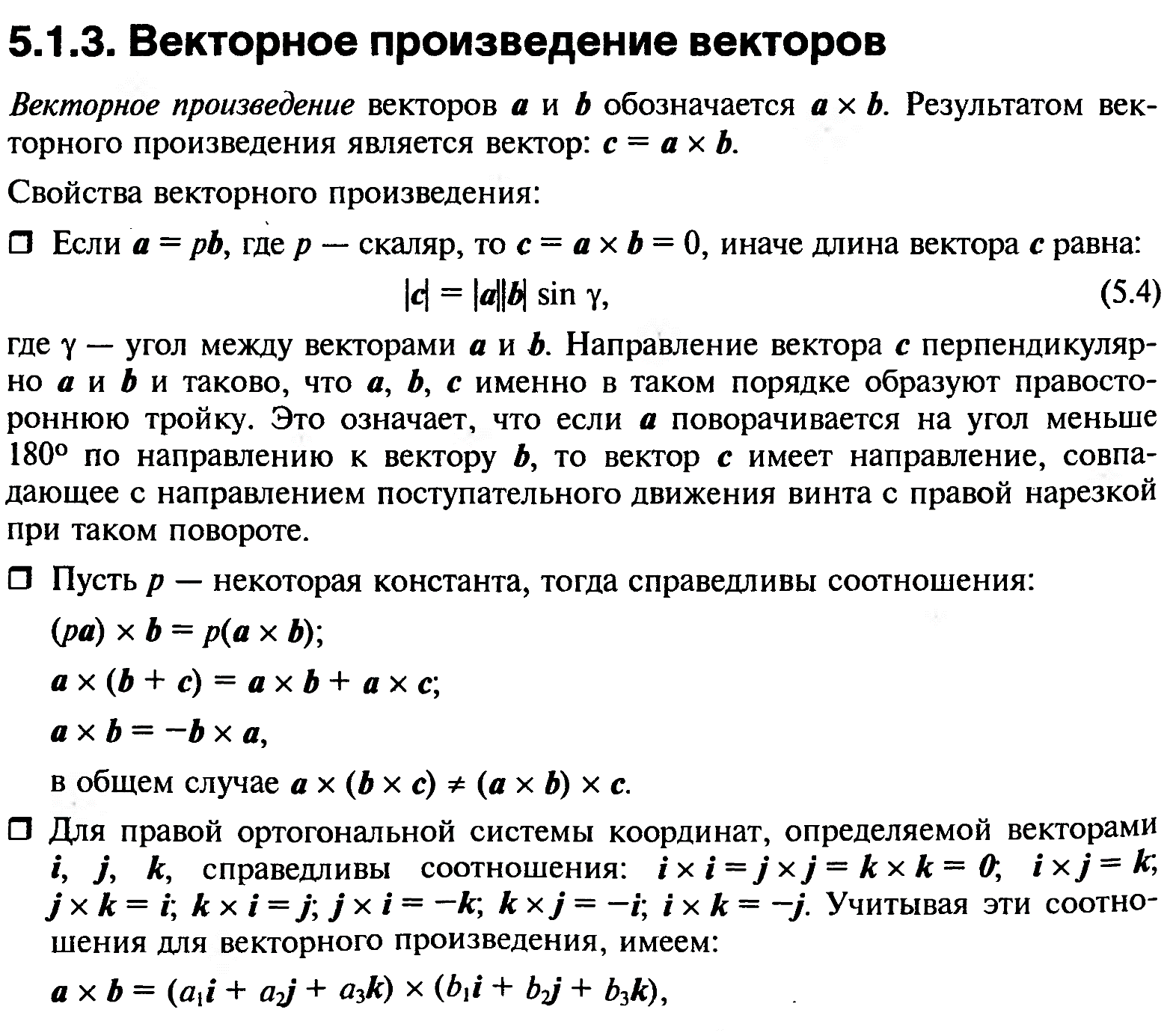
В прямоугольной системе координат  направление осей задается тройкой перпендикулярных единичных векторов.

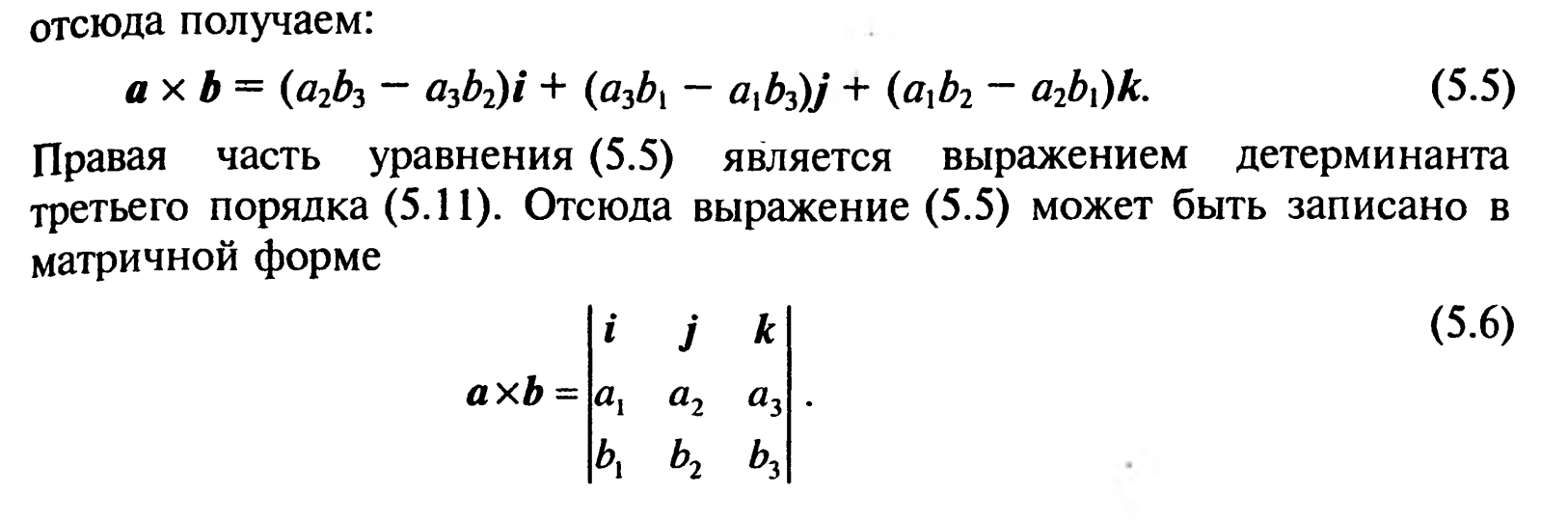
Система координат называется правой, если при повороте от вектора  к век­тору  на 90° направление вектора  совпадает с поступательным движением винта с правой резьбой (рис. ниже). Начальная точка векторов обозначается буквой О.











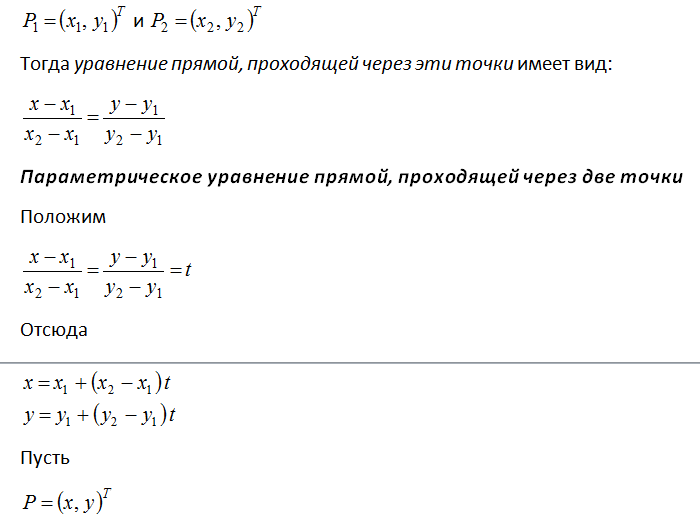
Полярная система координат

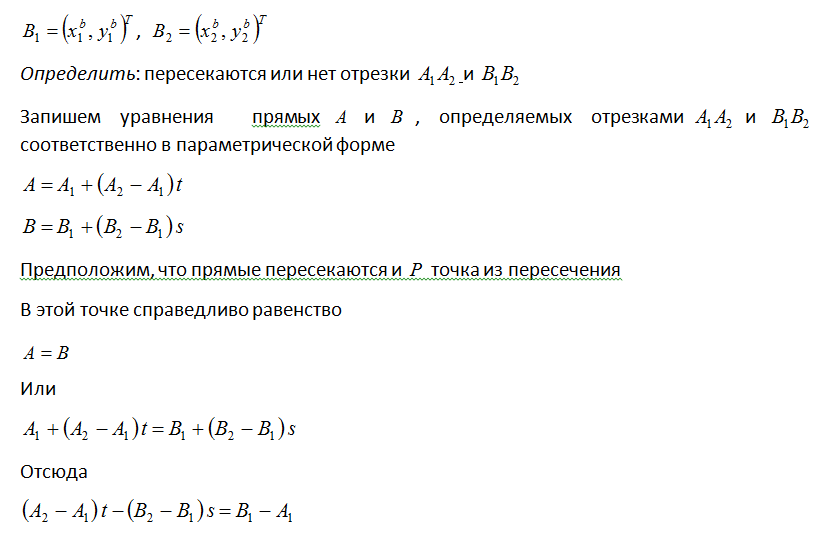
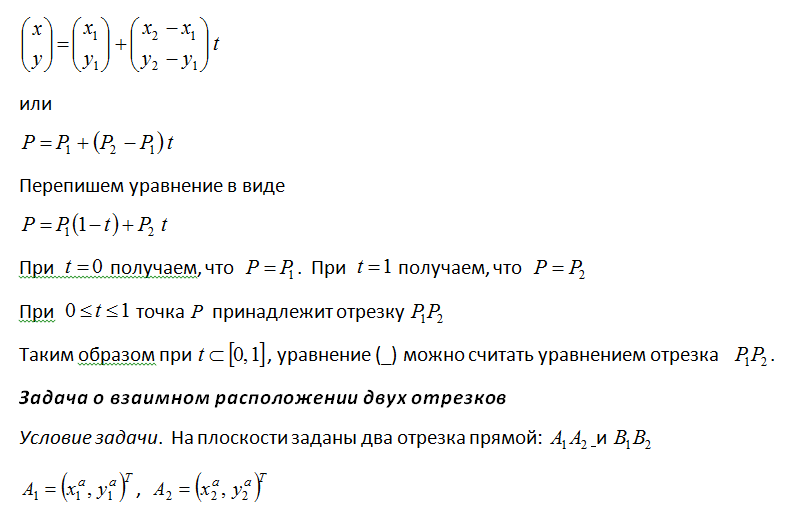
|  |  |
| --- | --- |
|  |  |

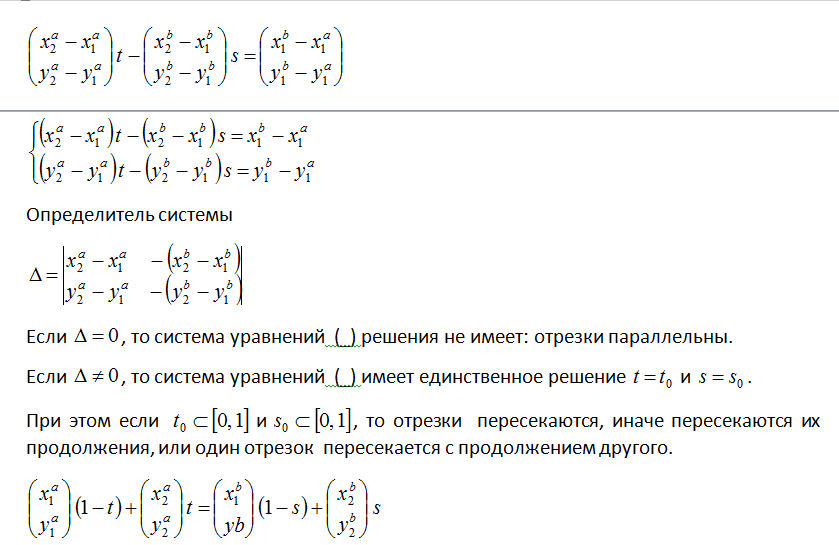
## 10. Геометрические основы компьютерной графики. Уравнение прямой, проходящей через две точки (на плоскости). Параметрическое уравнение прямой. Уравнение отрезка. Задача о взаимном расположении двух отрезков

Уравнение прямой, проходящей через две точки

Пусть в декартовой системе координат (ДСК) заданы две точки

Тогда можно записать

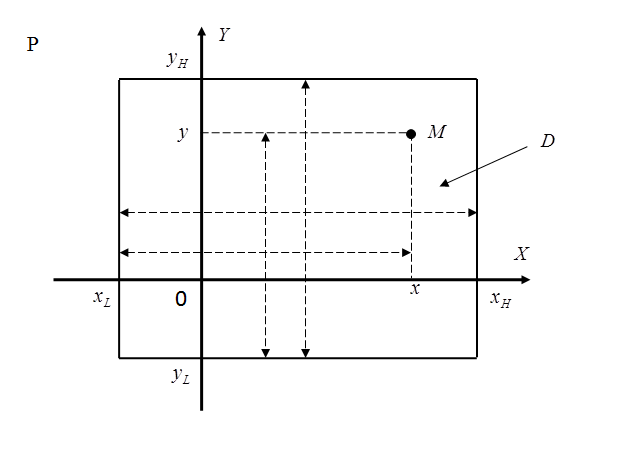


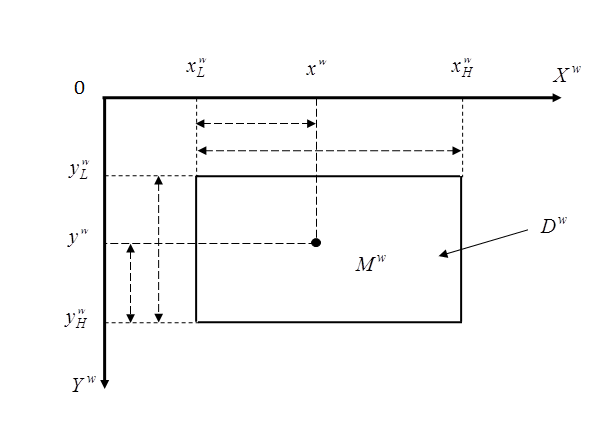


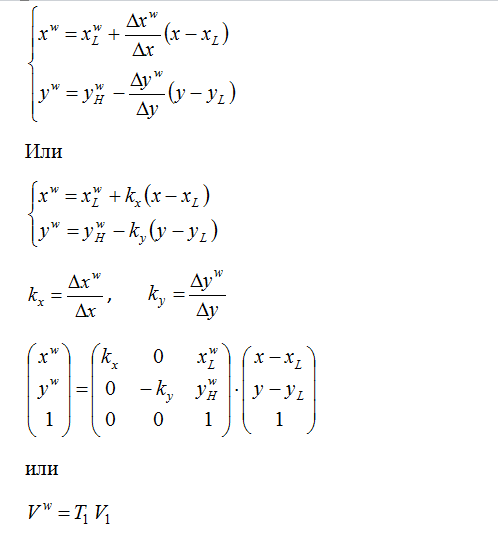
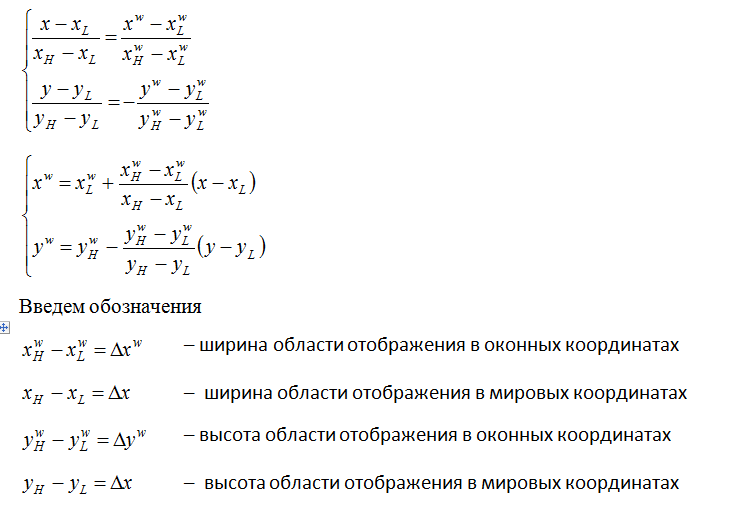
## 14. Мировые и экранные координаты. Алгоритм пересчёта мировых 2D-координат в оконные. Вывод и реализация

Мировые – система координат, которая вводится нужным образом в реальном мире и объекты описываются в данной системе координат**.**

Экранные – координаты на экране, всегда двумерные.







## 

## 15. Физическая и логическая системы координат. Режимы отображения (без настройки параметров). Алгоритм преобразования координат в GDI и его связь с алгоритмом пересчёта мировых координат в оконные

Физические координаты, как это следует из названия, имеют непосредственное отношение к физическому устройству вывода. В качестве единицы измерения длины в системе физических координат всегда используется пиксел. Если устройством вывода является экран монитора, физические координаты обычно называют экранными координатами.

Логические координаты передаются функциям GDI, выполняющим рисование фигур или вывод текста. Используемые единицы измерения зависят от режима отображения.

При отображении GDI преобразует логические координаты в физические. Способ преобразования зависит от режима отображения и других атрибутов контекста отображения, таких как расположение начала системы координат для окна, расположение начала системы физических координат, масштаб осей для окна и масштаб осей физических координат.

Физическая система координат

На рис. показана физическая система координат для экрана видеомонитора.

(0,0)

X

X

(x,y)

Начало этой системы координат располагается в левом верхнем углу экрана. Ось X направлена слева направо, ось Y - сверху вниз. В качестве единицы длины в данной системе координат используется пиксел.

Логическая система координат

Приложения Windows могут использовать одну из нескольких логических координат, устанавливая соответствующий режим отображения в контексте отображения. При этом, как мы уже говорили, можно использовать любое направление координатных осей и любое расположение начала координат. Например, возможна система координат, в которой задаются положительные и отрицательные координаты по любой оси.

Y

Y

(x,y)

Для установки режима отображения, непосредственно определяющего направление осей и размер логической единицы системы координат, используется функция SetMapMode:

int SetMapMode(HDC hdc, int nMapMode);

Параметр nMapMode может принимать одно из следующих значений.

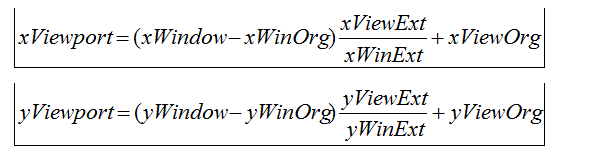
|  |  |  |  |
| --- | --- | --- | --- |
| ***Режим отображения*** | ***Направление оси X*** | ***Направление***  ***оси Y*** | ***Размер одной логической единицы*** |
| **MM\_TEXT** | Вправо | Вниз | 1 пиксел |
| **MM\_LOMETRIC** | Вправо | Вверх | 0,1 мм |
| **MM\_HIMETRIC** | Вправо | Вверх | 0,01 мм |
| **MM\_LOENGLISH** | Вправо | Вверх | 0,01 дюйм |
| **MM\_HIENGLISH** | Вправо | Вверх | 0,001 дюйм |
| **MM\_TWIPS** | Вправо | Вверх | 1/1440 дюйма |
| **MM\_ISOTROPIC** | Можно выбирать | Можно выбирать | Произвольный, одинаковый для осей X и Y |
| **MM\_ANISOTROPIC** | Можно выбирать | Можно выбирать | Произвольный, может быть разный для осей X и Y |

С помощью функции GetMapMode приложение может в любой момент времени определить номер режима отображения, выбранный в контекст отображения hdc:

int GetMapMode(HDC hdc);

Преобразование координат

Приложение, вызывая для рисования функции GDI, указывает логические координаты. Перед выводом GDI преобразует их в физические с использованием следующих формул:



Org – Origin – начало координат

В этих формулах используются следующие обозначения:

xWindow, yWindow – логические координаты по оси X и Y соответственно.

xViewport , yViewport – физические (экранные) координаты по оси X и Y соответственно.

Таким образом, логические координаты (xWindow,yWindow) преобразуются в физические координаты (xViewport,yViewport)**.**

xViewOrg, yViewOrg – определяют расположение начала физической системы координат, по умолчанию xViewOrg = 0 и yViewOrg = 0

Приложение может сместить начало физической системы координат, изменив значения переменных xViewOrg и yViewOrg.

xWinOrg, yWinOrg – определяют расположение начала логической системы координат, по умолчанию xWinOrg = 0 и yWinOrg = 0

Приложение может сместить начало логической системы координат, изменив значения переменных xWinOrg и yWinOrg.

xViewExt, xWinExt – задают масштаб, который используется в процессе преобразования координат по оси x.

yViewExt, yWinExt – задают масштаб, который используется в процессе преобразования координат по оси y.

Эти масштабы зависят от установленного режима отображения. Приложения могут изменить его только в режимах MM\_ISOTROPICи MM\_ANISOTROPIC , для остальных режимов отображения используются фиксированные значения.

## 16. Режимы отображения и настройка их параметров (функции класса CDC MFC). Функция *SetMyMode*, назначение, параметры и реализация

Для установки режима отображения, непосредственно определяющего направление осей и размер логической единицы системы координат, используется функция SetMapMode:

int SetMapMode(HDC hdc, int nMapMode);

Параметр nMapMode может принимать одно из следующих значений.

|  |  |  |  |
| --- | --- | --- | --- |
| **Режим отображения** | **Направление оси X** | **Направление**  **оси Y** | **Размер одной логической единицы** |
| MM\_TEXT | Вправо | Вниз | 1 пиксел |
| MM\_LOMETRIC | Вправо | Вверх | 0,1 мм |
| MM\_HIMETRIC | Вправо | Вверх | 0,01 мм |
| MM\_LOENGLISH | Вправо | Вверх | 0,01 дюйм |
| MM\_HIENGLISH | Вправо | Вверх | 0,001 дюйм |
| MM\_TWIPS | Вправо | Вверх | 1/1440 дюйма |
| MM\_ISOTROPIC | Можно выбирать | Можно выбирать | Произвольный, одинаковый для осей X и Y |
| MM\_ANISOTROPIC | Можно выбирать | Можно выбирать | Произвольный, может быть разный для осей X и Y |

Режим MM\_TEXT

Режим отображения MM\_TEXT устанавливается в контексте отображения по умолчанию. Для этого режима формулы преобразования координат упрощаются:

xViewport = (xWindow - xWinOrg) + xViewOrg

yViewport = (yWindow - yWinOrg) + yViewOrg

Так как по умолчанию xViewOrg=0, yViewOrg=0, xWinOrg=0, yWinOrg=0, то

xViewport = xWindow

yViewport = yWindow

Соответствующая система координат представлена на рис. ниже (начало системы координат расположено точно в левом верхнем углу внутренней области окна, рисунок иллюстрирует только направление координатных осей).

Y

X

(x,y)

Так как в формуле преобразования не присутствуют переменные xViewExt, yViewExt, xWinExt и yWinExt, в данном режиме преобразования невозможно изменить масштаб осей координат. Поэтому логическая единица длины в режиме отображения MM\_TEXT равна физической, т. е. одному пикселу.

Тем не менее, приложение может изменить смещение физической или логической системы координат, изменив, соответственно, значение пар переменных (xViewOrg, yViewOrg) и (xWinOrg,yWinOrg). Для установки смещения можно использовать функции SetViewportOrgEx и SetWindowOrgEx.

BOOL SetViewportOrgEx(

HDC hdc, // контекст отображения

int nXOrigin, // новое значение для xWinOrg

int nYOrigin, // новое значение для yWinOrg

LPPOINTlpPoint // указатель на структуру POINT

);

**virtual CPoint SetViewportOrg(** **int** *x*, **int** *y* **); //** Метод класса CDC MFC

**CPoint SetViewportOrg(POINT** *point* **); //** Метод класса CDC MFC

**CPoint SetViewportOrg(CPoint** *point* **); //** POINT или CPoint

BOOL SetWindowOrgEx(

HDC hdc, // контекст отображения

int nXOrigin, // новое значение для xWinOrg

int nYOrigin, // новое значение для yWinOrg

LPPOINTlpPoint // указатель на структуру POINT

);

**CPoint SetWindowOrg(int** *x*, int *y* **); //** Метод класса CDC MFC

**CPoint SetWindowOrg( POINT** *point* **); //** Метод класса CDC MFC

**CPoint SetWindowOrg( CPoint** *point* **); //** POINT или CPoint

В структуру, адрес которой передается через параметр lpPoint, записываются старые координаты начала системы координат. Если они не нужны, можно положить lpPoint=NULL.

Обе функции возвращают TRUE в случае успеха и FALSE при возникновении ошибки.

В любой момент времени можно определить расположение начала физических или логических координат, если воспользоваться функциями GetViewportOrgEx и GetWindowOrgEx.

Метрические режимы отображения

Режим MM\_LOMETRIC, наряду с режимами MM\_HIMETRIC, MM\_LOENGLISH, MM\_HIENGLISH и MM\_TWIPS, относится к метрическим режимам. Эти режимы отображения позволяют использовать привычные единицы измерения, такие как миллиметры и дюймы.

В метрических режимах отображения используются полные формулы преобразования координат, приведенные выше в разделе "Преобразование координат". В этих формулах приложение может изменять переменные, определяющие смещение начала физической или логической системы координат xViewOrg, yViewOrg, xWinOrg и yWinOrg.

Приложение не может изменить значения переменных xViewExt, yViewExt, xWinExt и yWinExt, от которых зависит масштаб по осям координат. Отношения xViewExt/xWinExt и yViewExt/yWinExt имеют фиксированное значение для каждого из метрических режимов отображения.

Заметим, что для этих режимов отношение yViewExt/yWinExt имеет отрицательный знак, в результате чего ось Y оказывается направленной снизу вверх.

-------------------------------------------------------------------------------------------------------------------

Обратим внимание на одно важное обстоятельство, связанное с использованием метрических режимов отображения.

Сразу после переключения в метрический режим отображения система координат примет достаточно странный вид (рис. ниже).

Y(-)

X(+)

(x,y)

(0,0)

Ось X, как и следовало ожидать, окажется направленной слева направо, а ось Y - снизу вверх. Точка с координатами (0,0) будет находиться в верхнем левом углу экрана, поэтому для того чтобы нарисовать что-нибудь в такой системе координат, вам придется для y-координаты графических объектов использовать отрицательные числа. Для того чтобы система координат приняла более удобный вид, можно переместить начало физических координат в нижний левый угол окна или в центр окна.

Прежде, чем выполнять перемещение начала координат, следует определить размеры внутренней области окна. Это можно сделать при обработке сообщения WM\_SIZE:

---------------------------------- Для MFC -----------------------------------------------------

int dxClient, dyClient; // Объявлены в классе CChildView

void CMainFrame::OnSize(UINT nType, int cx, int cy)

{

CFrameWnd::OnSize(nType, cx, cy);

m\_wndView.dxClient=cx; // m\_wndView - переменная типа CChildView

m\_wndView.dyClient=cy;

}

---------------------- Для API --------------------------------------------------------------------

static short cxClient, cyClient;

....

case WM\_SIZE:

{

cxClient = LOWORD(lParam);

cyClient = HIWORD(lParam);

....

return 0;

}

-------------------------------------------------------------------------------------------------------

Для того чтобы расположить начало координат в левом нижнем углу окна, следует вызвать функцию SetViewportOrgEx, передав ей новые координаты начала физической системы координат (0,cyClient):

SetViewportOrgEx(hdc, 0, cyClient, NULL);

Полученная в результате система координат показана на рис.:

Y(+)

X(+)

(0,0)

Аналогичным образом можно расположить начало системы координат в середине окна (рис.6), обеспечив возможность использования положительных и отрицательных координат вдоль оси X и Y:

SetViewportOrgEx(hdc, cxClient/2, cyClient/2,NULL);

Y(+)

X(+)

(0,0)

Пример

void CChildView::OnPaint()

{

CPaintDC dc(this); // device context for painting

if(Index==1)

{

dc.SetMapMode(MM\_LOMETRIC); // 0,1 мм

dc.SetViewportOrg(0,dyClient);

//SetViewportOrgEx(hdc, 0, dyClient, NULL); //API

dc.MoveTo(0,0);

dc.LineTo(1000,1000);

}

if(Index==2)

{

dc.SetMapMode(MM\_LOMETRIC); // 0,1 мм

dc.SetViewportOrg(dxClient/2,dyClient/2);

//SetViewportOrgEx(hdc, , dyClient/2,NULL); //API

dc.MoveTo(-500,0);

dc.LineTo(500,0); //Ось X

dc.MoveTo(0,-500);

dc.LineTo(0,500); //Ось Y

dc.MoveTo(0,0);

dc.LineTo(200,200);

}

}

Режимы MM\_ISOTROPIC и MM\_ANISOTROPIC

Режимы отображения MM\_ISOTROPIC (изотропный) и MM\_ANISOTROPIC (анизотропный) допускают изменение направления осей X и Y, а также изменение масштаба осей координат. В изотропном режиме отображения MM\_ISOTROPIC масштаб вдоль осей X и Y всегда одинаковый (т. е. для обоих осей используются одинаковые логические единицы длины). Анизотропный режим MM\_ANISOTROPIC предполагает использование разных масштабов для разных осей (хотя можно использовать и одинаковые масштабы).

Для изменения ориентации и масштаба осей предназначены функции SetViewportExtEx и SetWindowExtEx.

Функция SetWindowExtEx устанавливает для формулы преобразования координат значения переменных xWinExt и yWinExt:

BOOL SetViewportExtEx(

HDC hdc, // идентификатор контекста отображения

int nXExtent, // значение для xViewExt

int nYExtent, // значение для yViewExt

LPSIZE lpSize // указатель на структуру SIZE

);

**virtual CSize SetViewportExt(int** *cx*, **int** *cy* **); //** Метод класса CDC MFC

**CSize SetViewportExt( SIZE** *size* **); //** Метод класса CDC MFC

Функция SetViewportExtEx должна использоваться после функции SetWindowExtEx. Она устанавливает для формулы преобразования координат значения переменных xViewExt и yViewExt:

BOOL SetWindowExtEx(

HDC hdc, // идентификатор контекста отображения

int nXExtent, // значение для xWinExt

int nYExtent, // значение для yWinExt

LPSIZE lpSize // указатель на структуру SIZE

);

**virtual CSize SetWindowExt( int** *cx***, int** *cy* **); //** Метод класса CDC MFC

**CSize SetWindowExt(SIZE** *size* **); //** Метод класса CDC MFC

Старые значения переменных, определяющих масштаб преобразования, записываются в структуру SIZE, указатель на которую передается через параметр lpSize.

typedef struct tagSIZE {

LONG cx;

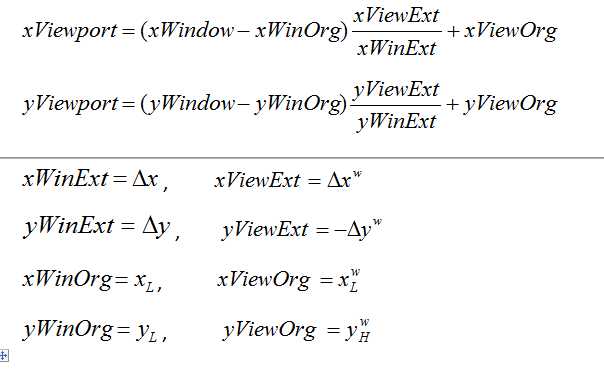
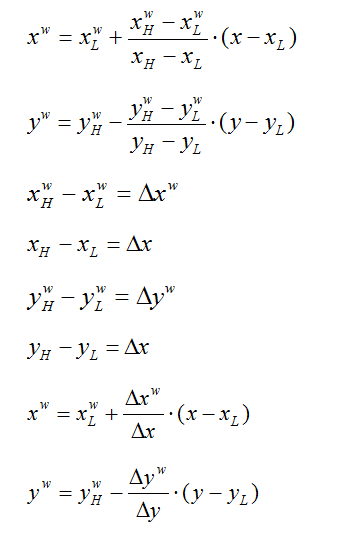
LONG cy;

} SIZE, \*PSIZE;

Если они не нужны, можно положить lpSise=**NULL**

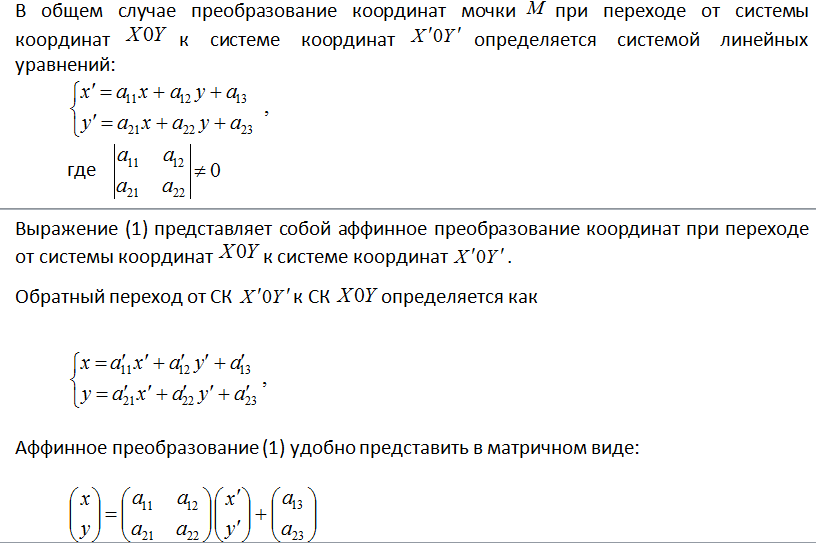
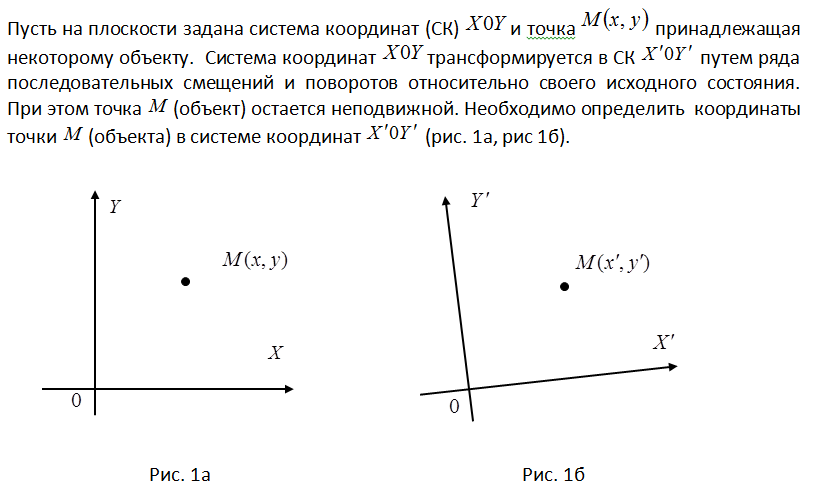
Изотропный режим отображения удобно использовать в тех случаях, когда надо сохранить установленное отношение масштабов осей X и Y при любом изменении размеров окна, в которое выводится изображение.

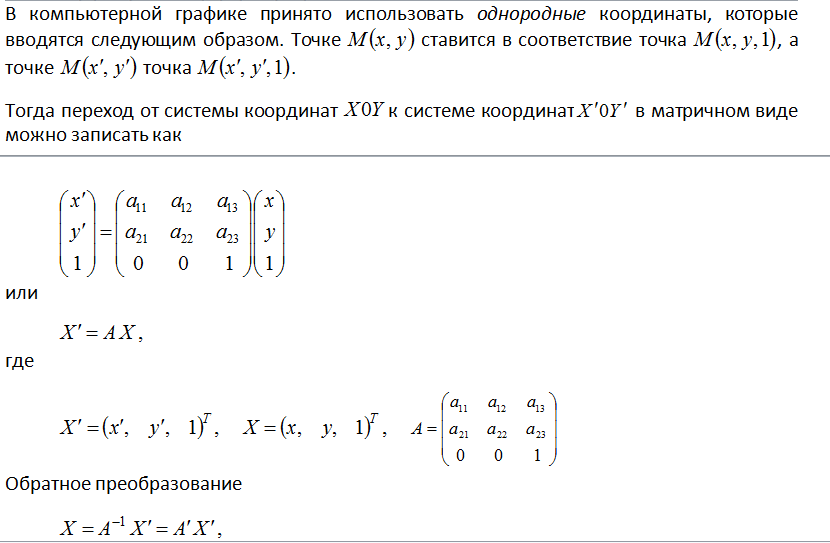
Анизотропный режим удобен в тех случаях, когда изображение должно занимать всю внутреннюю поверхность окна при любом изменении размеров окна. Соотношение масштабов при этом не сохраняется.

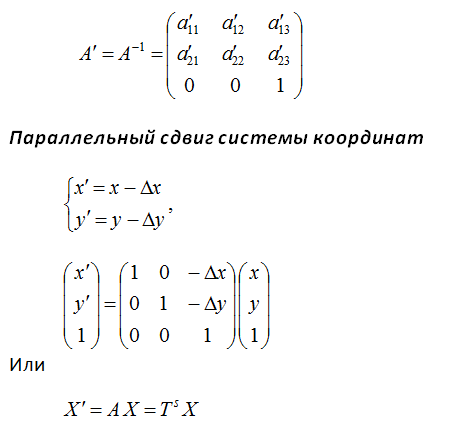


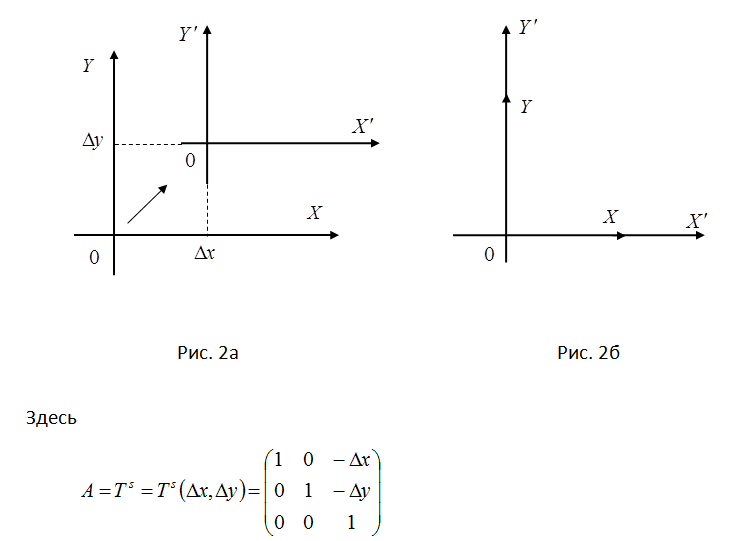
|  |  |
| --- | --- |
| void SetMyMode(HDC hdc, RECT RS,  RECT RW) // API  // hdc - контекст устройства  // RS - область в мировых координатах  // RW - область в оконных координатах  {  int dsx=RS.right-RS.left;  int dsy=RS.top-RS.bottom;  int xsL=RS.left;  int ysL=RS.bottom;  int dwx=RW.right-RW.left;  int dwy=RW.bottom-RW.top;  int xwL=RW.left;  int ywH=RW.bottom;  SetMapMode(hdc,MM\_ANISOTROPIC);  SetWindowExtEx(hdc,dsx, dsy, NULL);  SetViewportExtEx(hdc, dwx,-dwy, NULL);  SetWindowOrgEx (hdc, xsL, ysL, NULL);  SetViewportOrgEx(hdc,xwL, ywH, NULL);  } | void SetMyMode(CDC& dc, CRect& RS,  CRect& RW) // MFC  // dc - ссылка на класс CDC MFC  // RS - область в мировых координатах  // RW - область в оконных координатах  {  int dsx=RS.right-RS.left;  int dsy=RS.top-RS.bottom;  int xsL=RS.left;  int ysL=RS.bottom;  int dwx=RW.right-RW.left;  int dwy=RW.bottom-RW.top;  int xwL=RW.left;  int ywH=RW.bottom;  dc.SetMapMode(MM\_ANISOTROPIC);  dc.SetWindowExt(dsx,dsy);  dc.SetViewportExt(dwx,-dwy);  dc.SetWindowOrg(xsL,ysL);  dc.SetViewportOrg(xwL,ywH);  } |

## 17. Аффинные преобразования на плоскости. Преобразование системы координат: смещение, растяжение-сжатие, поворот

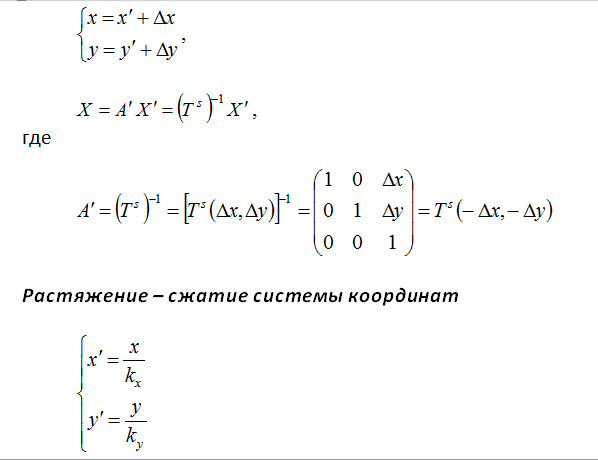


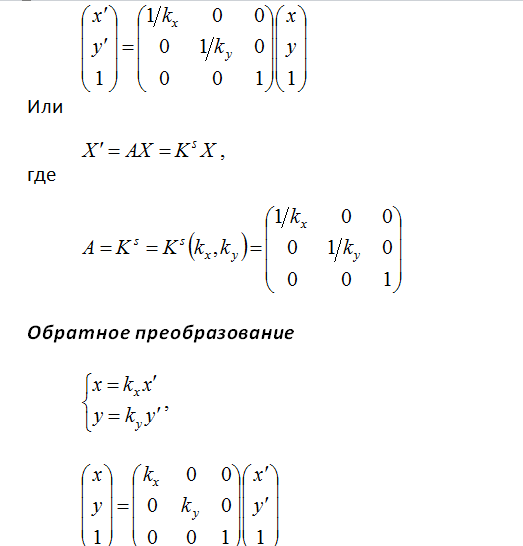
где



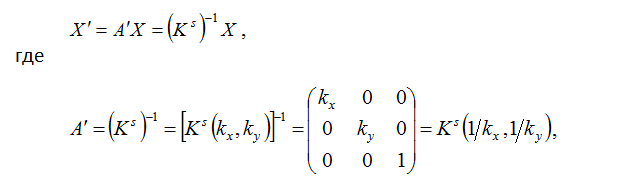


Обратное преобразование

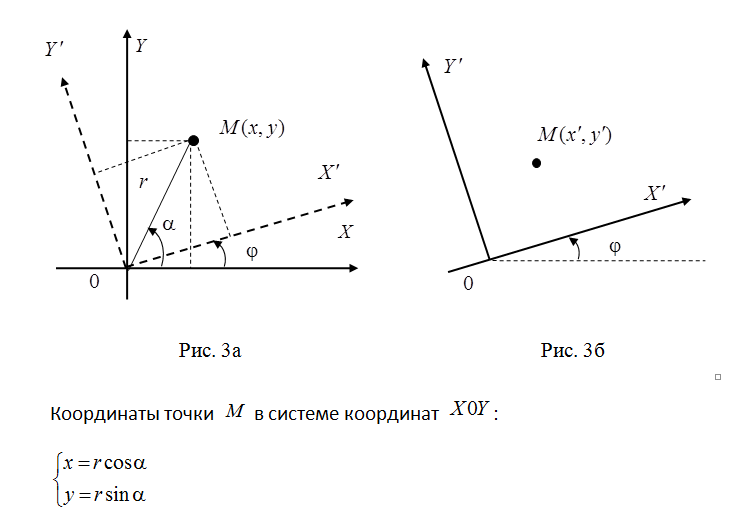


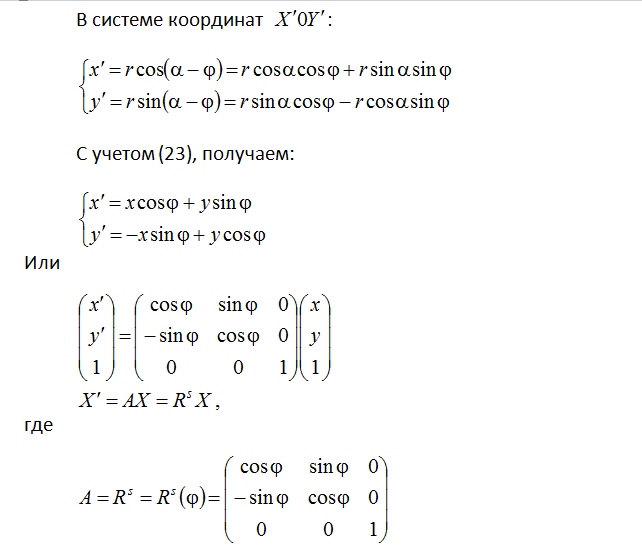


Или



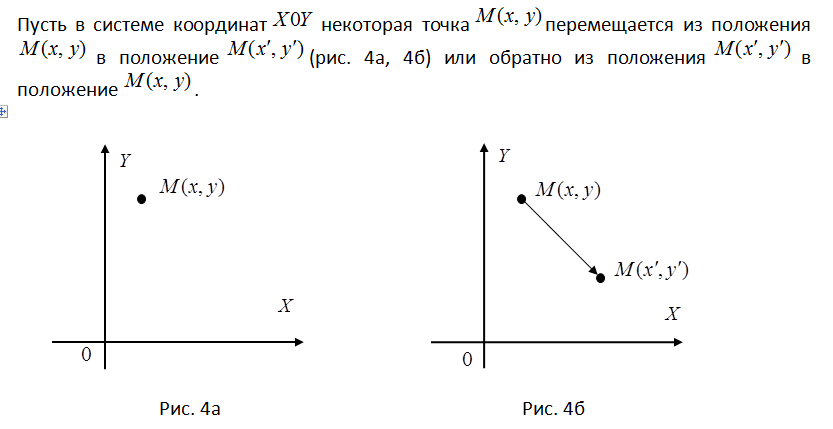
Поворот системы координат



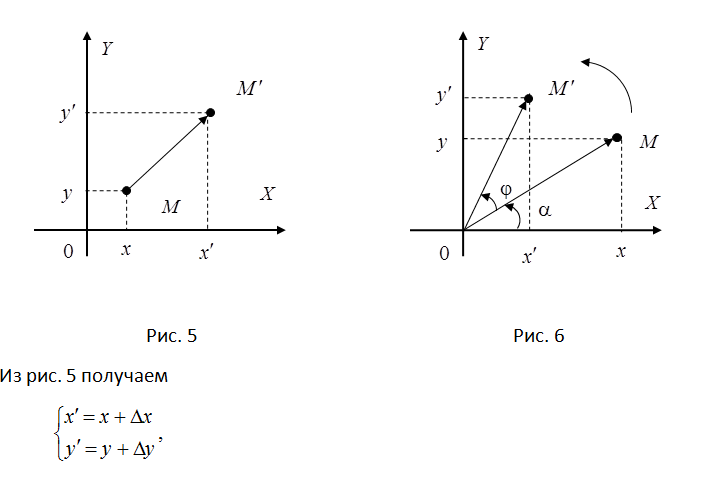
******

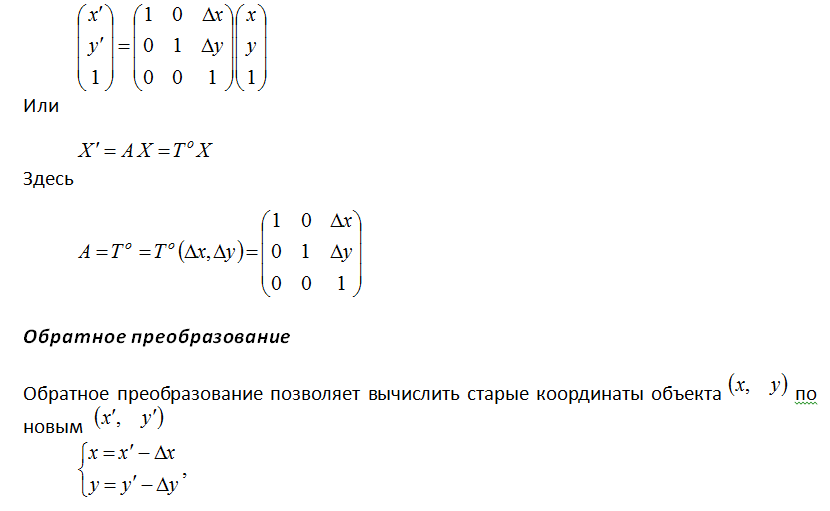
## 18. Аффинные преобразования на плоскости. Преобразование объектов: смещение, растяжение-сжатие, поворот

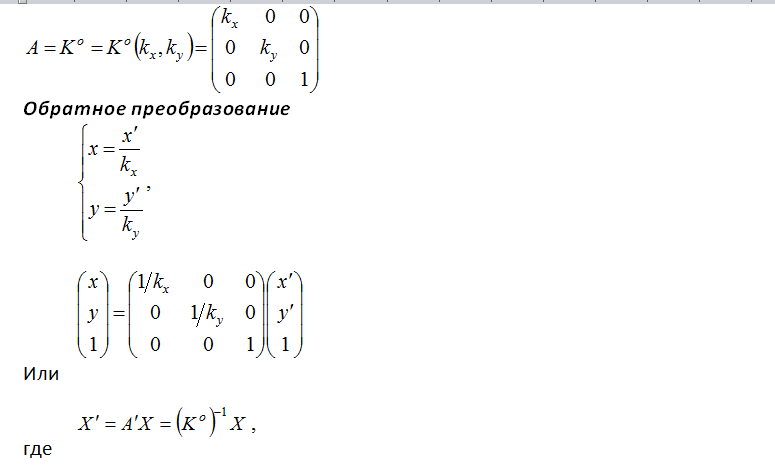
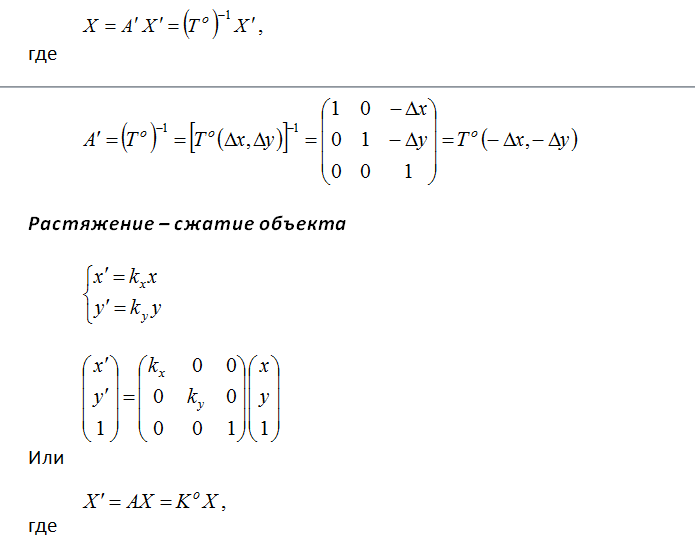
Под преобразованием объектов будем понимать изменение координат точек, принадлежащих этому объекту при изменении его положения в некоторой системе координат.

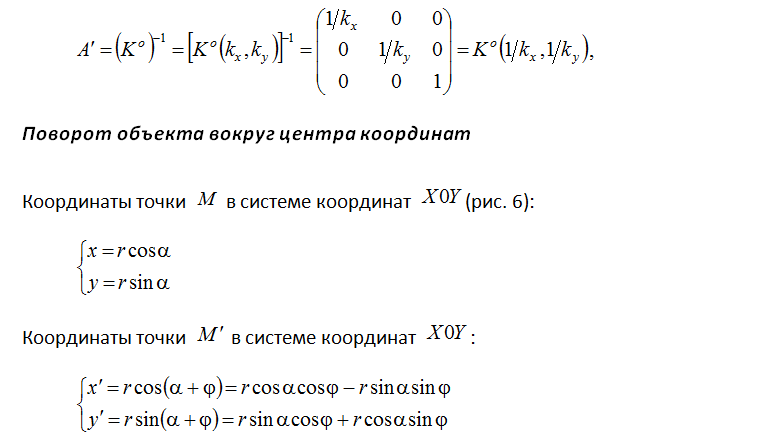
Рассмотрим частные случаи аффинных преобразований объектов.

Сдвиг объекта

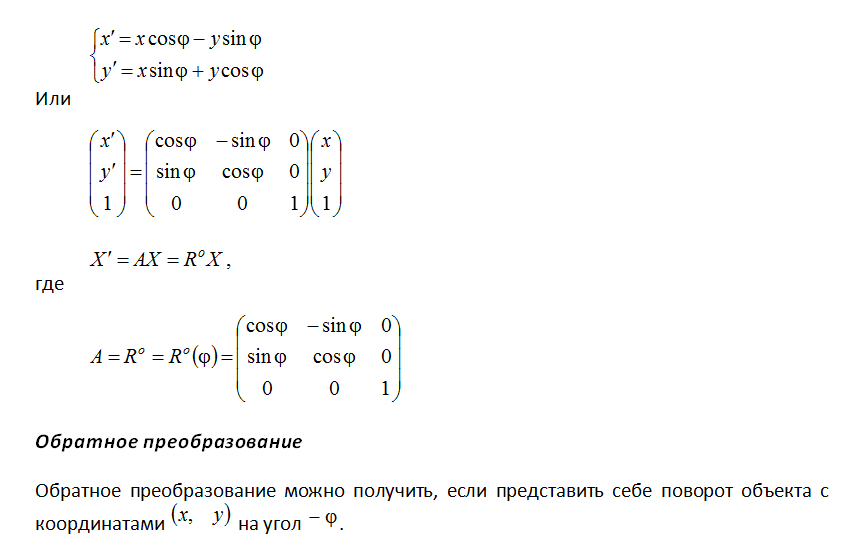


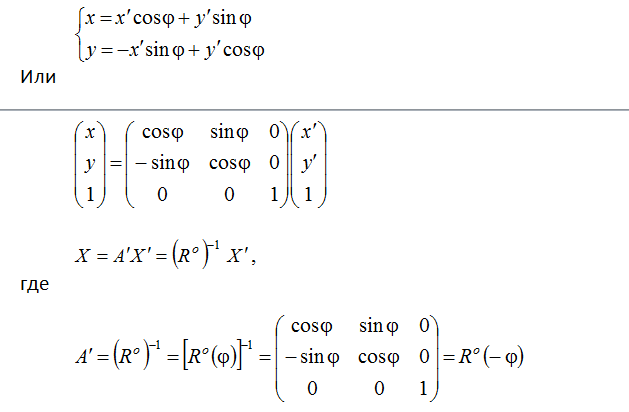






С учетом (46), получаем:



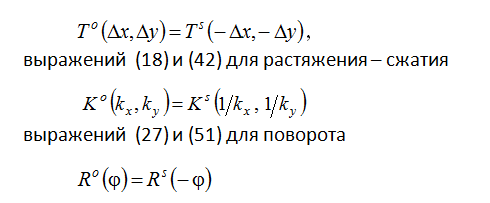


Связь преобразований объектов с преобразованиями координат

Преобразование объектов и преобразование систем координат тесно связаны между собой. Движение объектов можно рассматривать как движение в обратном направлении соответствующей системы координат.

Такая относительность движения дает дополнительные возможности для моделирования и визуализации различных объектов.

Запишем соответствующие соотношения, основываясь на сравнении выражений (11) и (35) для сдвига



**CMatrix CreateTranslate2D(double dx, double dy);**

// Формирует матрицу для преобразования координат ОБЪЕКТА при

// его смещении на dx по оси X и на dy по оси Y в фиксированной системе

// координат

// --- ИЛИ ---

// Формирует матрицу для преобразования координат объекта при

// смещении начала системы координат на -dx оси X и на -dy по оси Y при

// фиксированном положении объекта

**CMatrix CreateRotate2D(double fi);**

// Формирует матрицу для преобразования координат ОБЪЕКТА при его повороте

// на угол fi (при fi>0 против часовой стрелки)в фиксированной системе координат

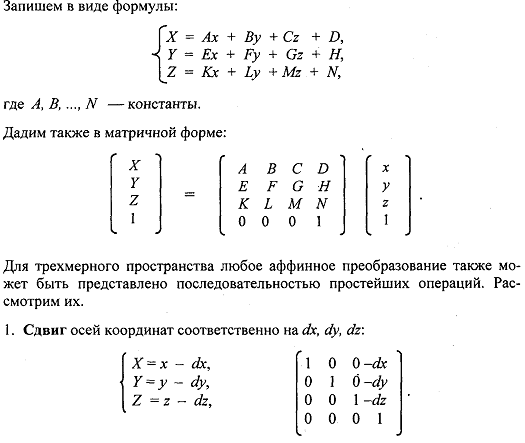
// --- ИЛИ ---

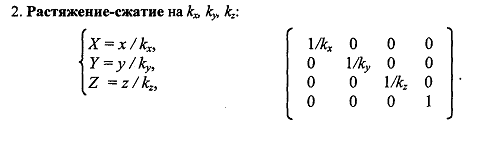
// Формирует матрицу для преобразования координат объекта при повороте начала

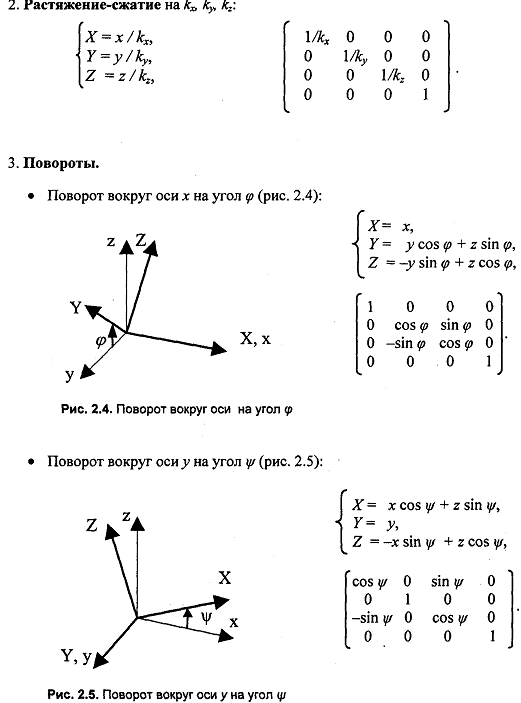
// системы координат на угол -fi при фиксированном положении объекта

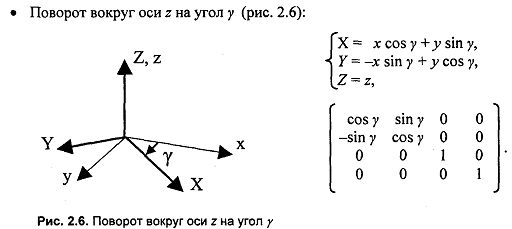
// fi - угол ***в градусах***

## 19. Аффинные преобразования в пространстве. Преобразование системы координат: смещение, растяжение-сжатие, повороты вокруг осей координат

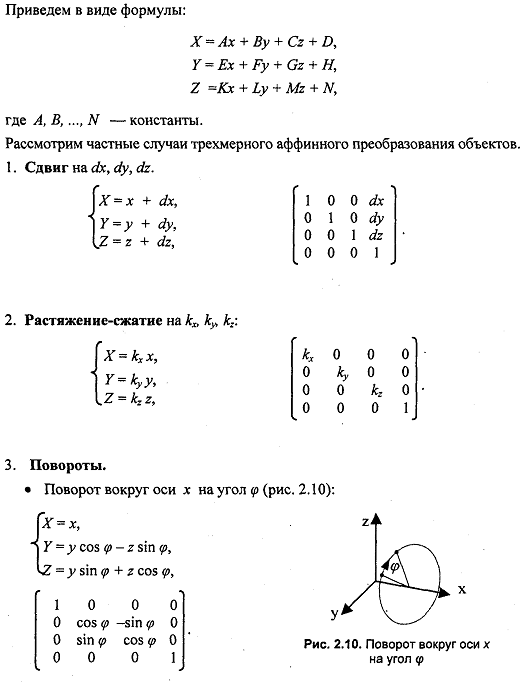


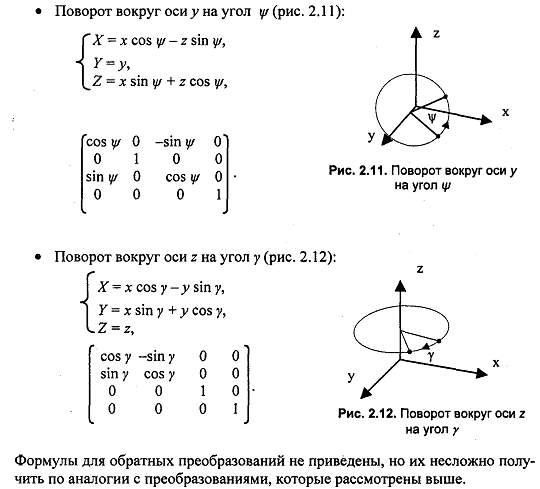






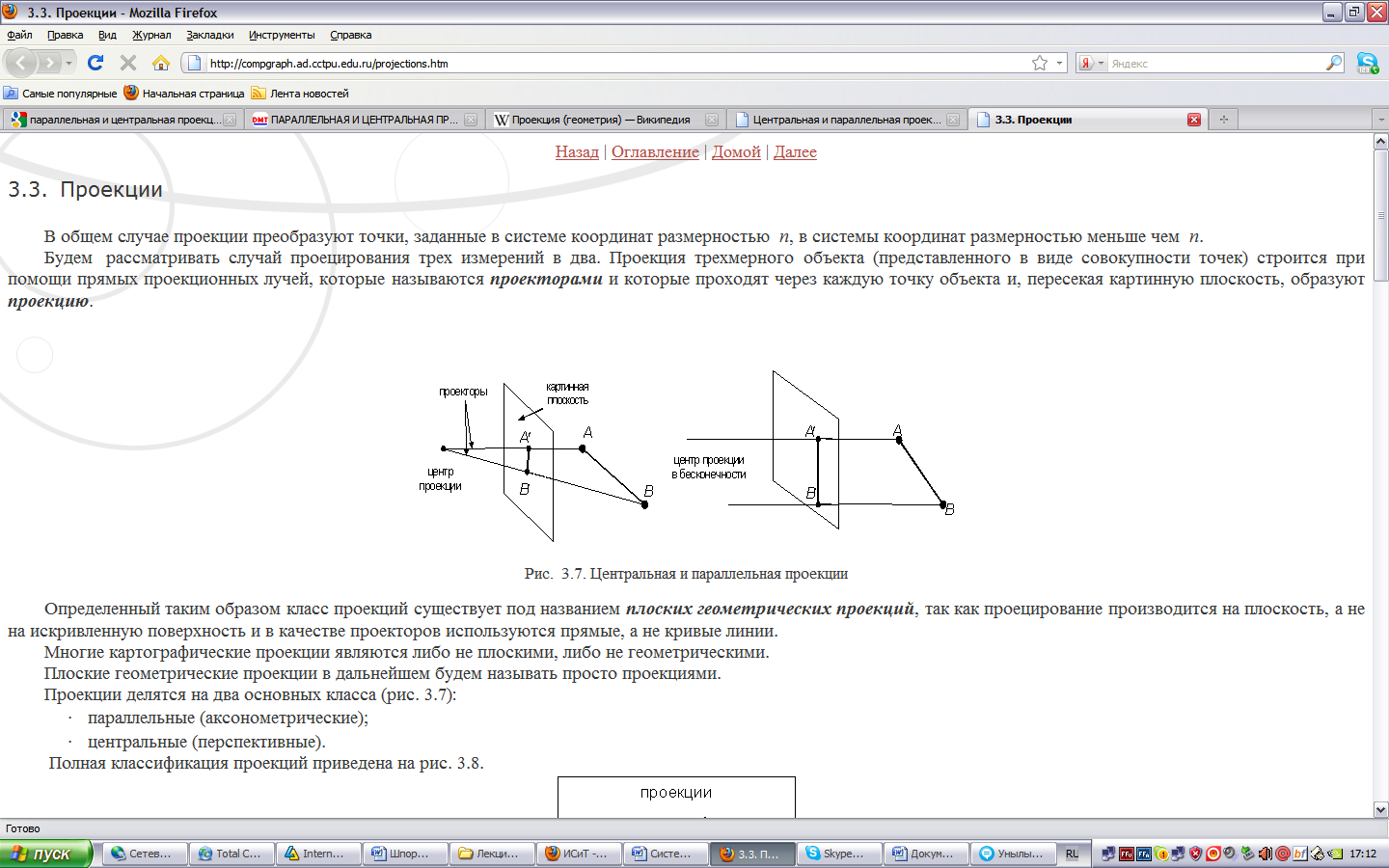
## 20. Аффинные преобразования в пространстве. Преобразования объектов: смещение, растяжение-сжатие, повороты вокруг осей координат



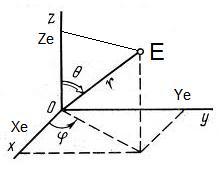


## 21. Основные типы проекций. Видовая система координат (вывод матрицы преобразования)

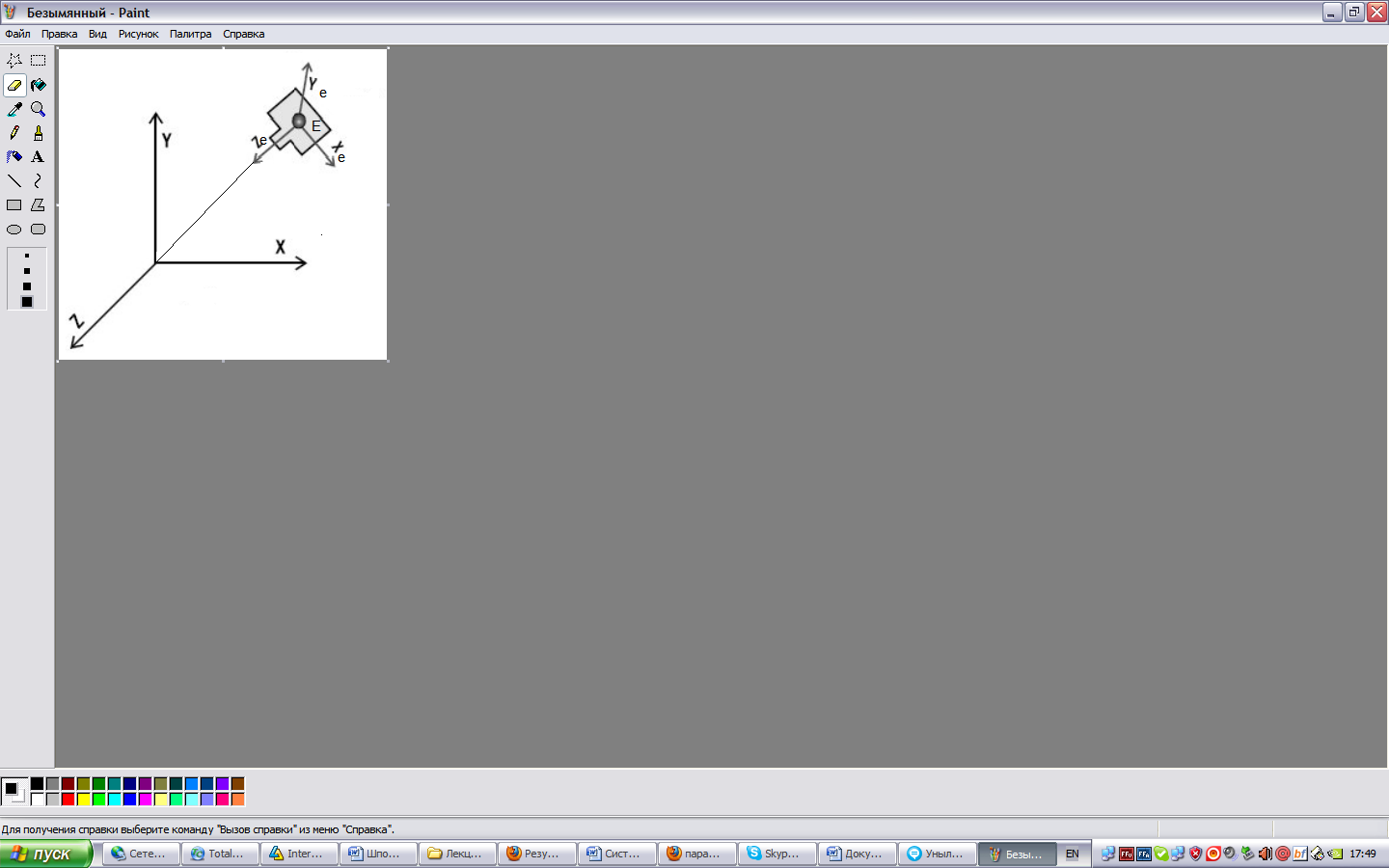
Картинная плоскость-плоскость, на которой выполняется проекция объекта. В компьютерной графике наиболее распространены центральные (рис. слева) и параллельные (рис. справа) проекции.

****

Видовая система координат

Для изображения объекта на экране необходимо преобразовать его координаты в некую другую СК, которая связана с точкой наблюдателя (положением камеры). Эта СК называется видовой. Мировая система координат традиционно является правосторонней, а видовая систем координат левосторонней.

Рассмотрим правую систему мировых координат. С помощью сферических координат зададим положение точки наблюдателя.

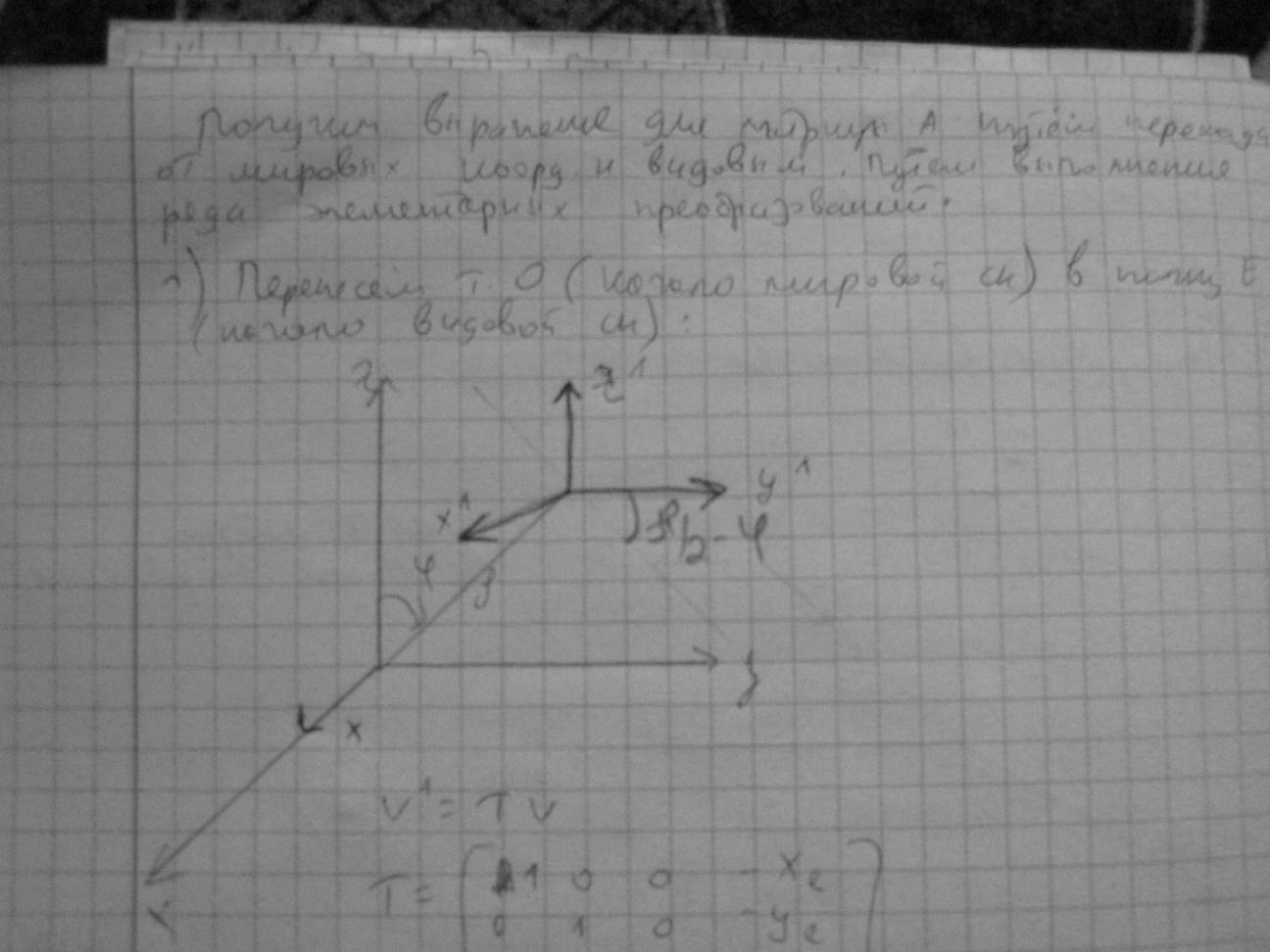
Изобразим видовую СК:

Ось Z смотрит в начало мировой СК. V(видовая)=A\*V(мировая)

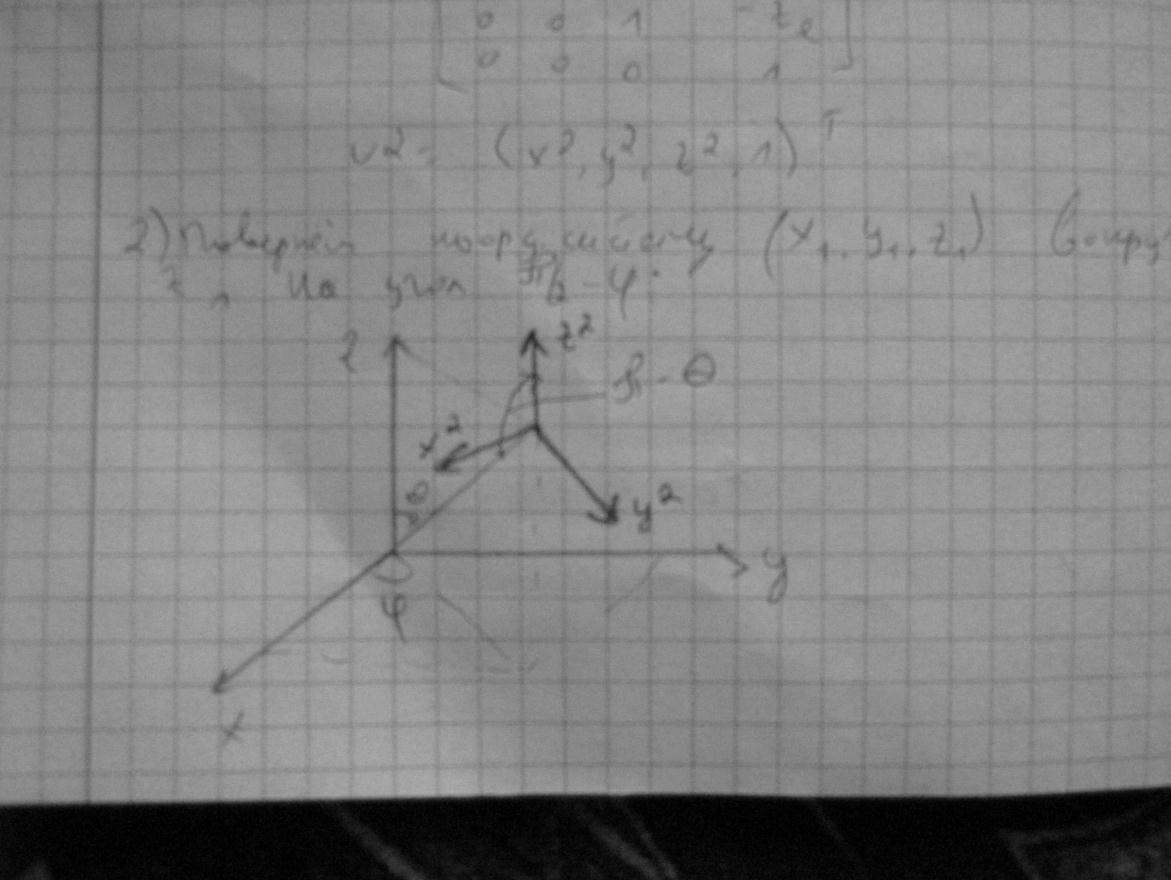
Получим выражение для матрицы А путем перехода от мировых координат к видовым. Путем выполнения ряда элементарных преобразований.

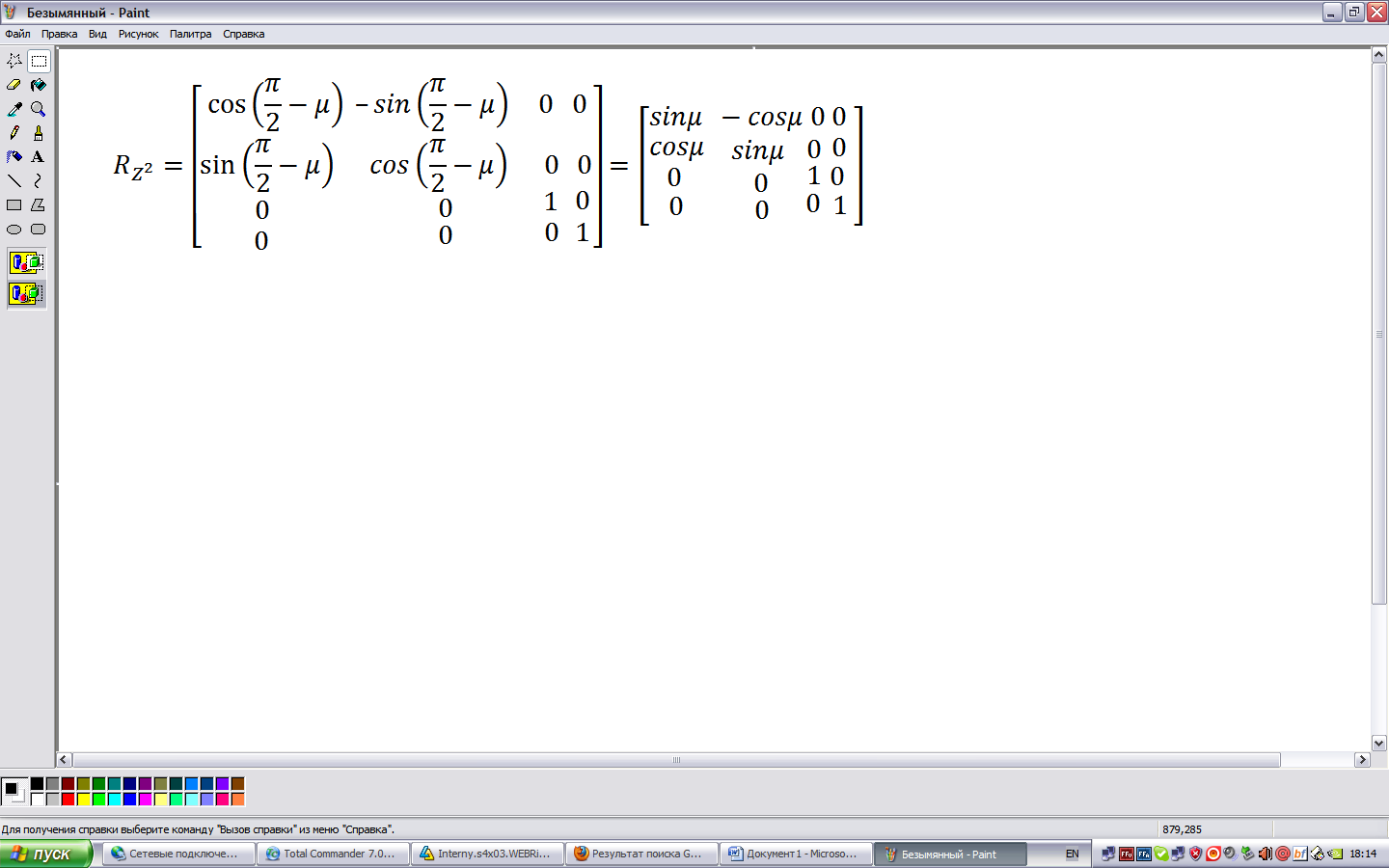
Перенесем точку О (начало мировой СК) в точку Е (начало видовой СК)

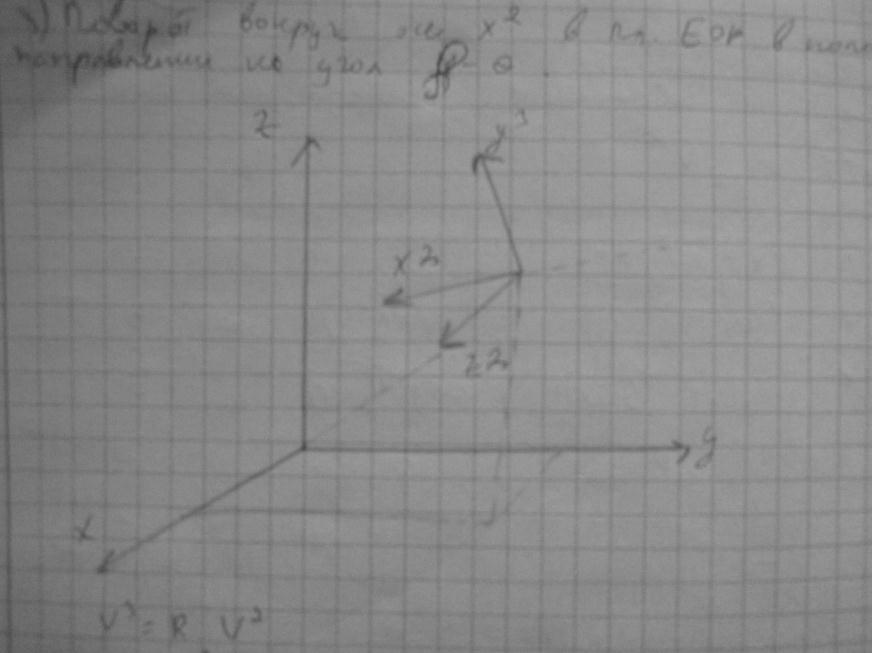
V^1=TV

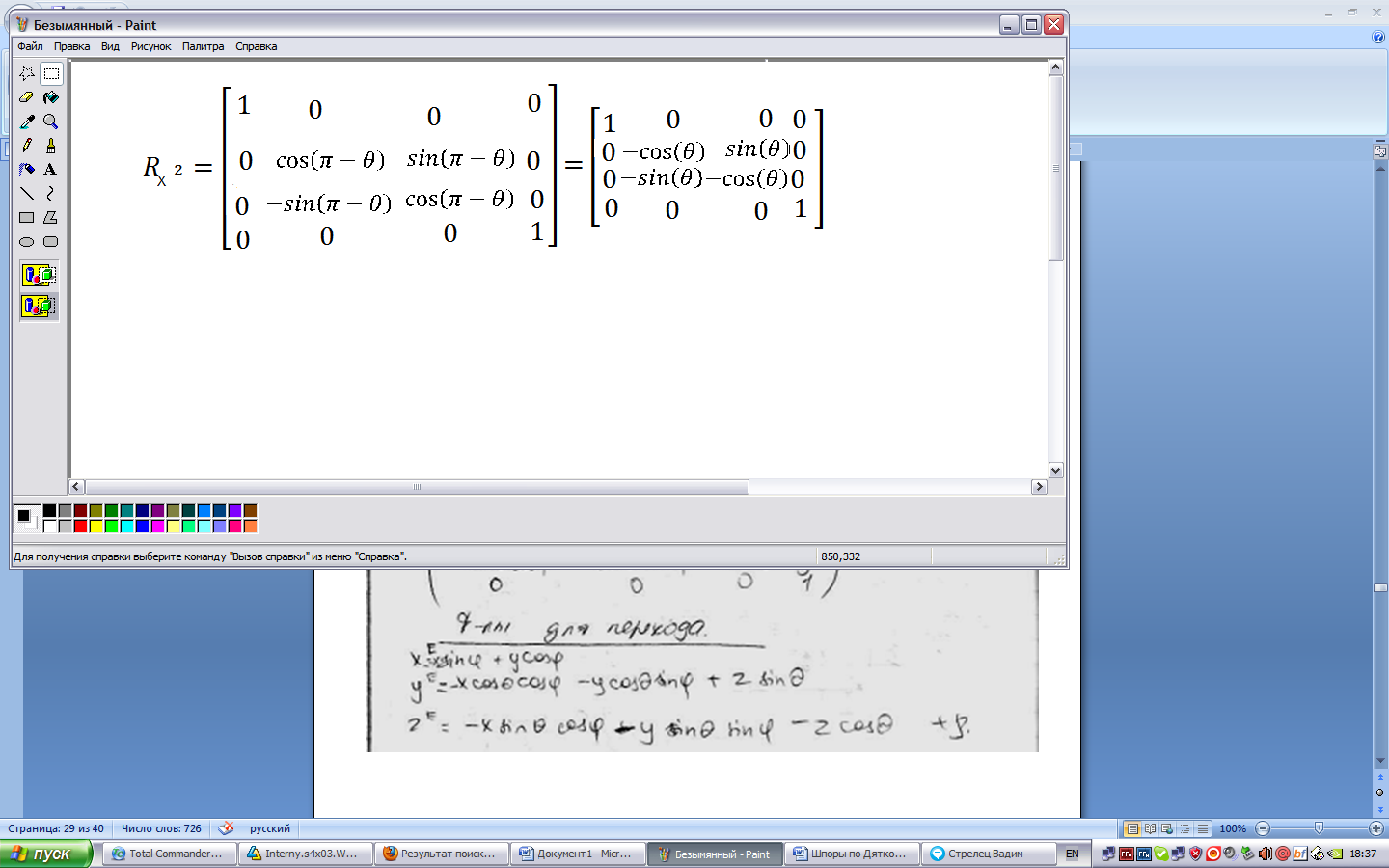


Повернем координатную систему (X1,Y1,Z1) вокруг оси Z1 на угол π/2-µ. Поворот выполняем в отрицательном направлении (против часовой стрелки) => матрица данного преобразования будет совпадать с преобразованием точки.

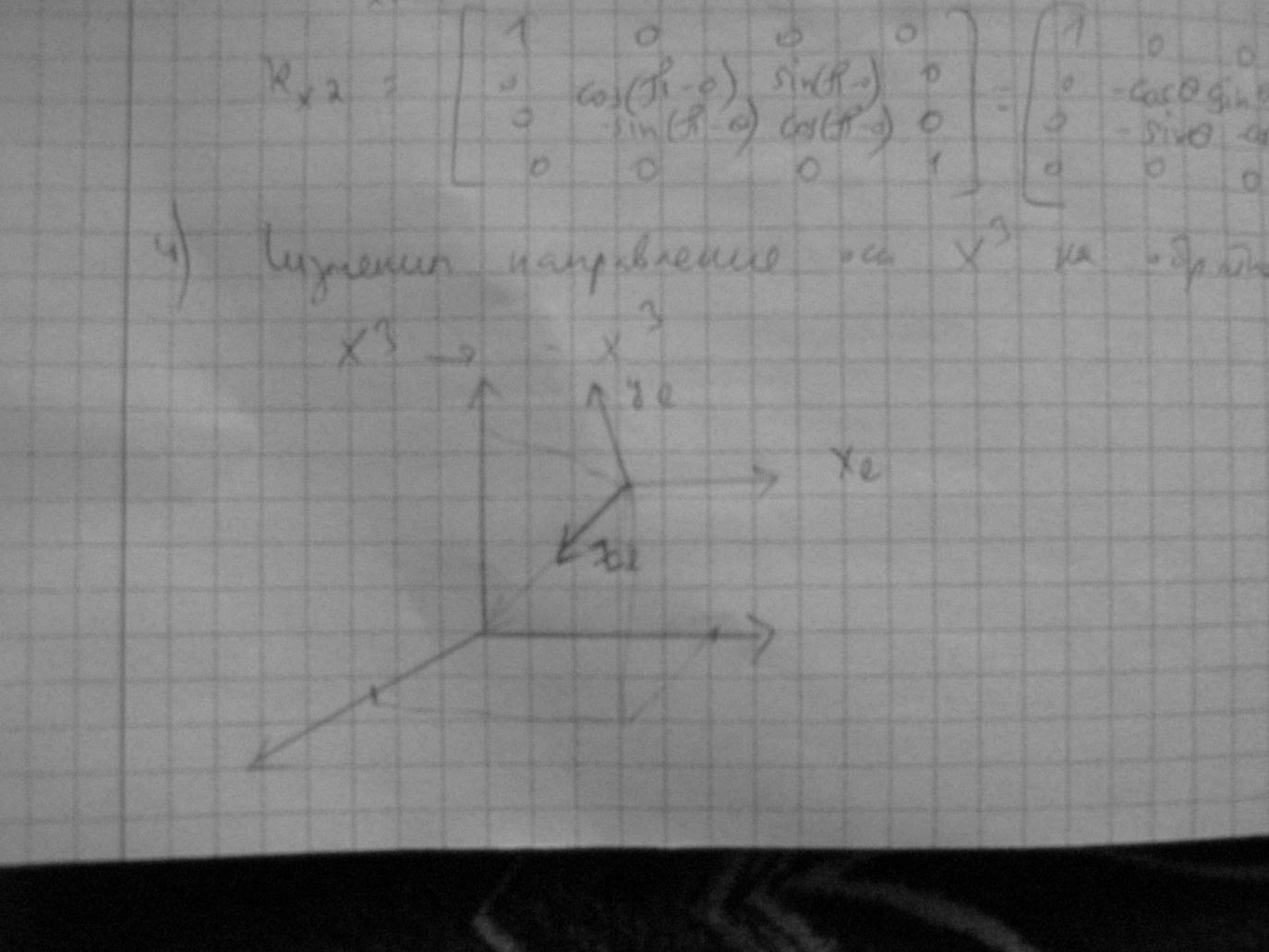
 ;

**

Поворот вокруг оси в положительном направлении на угол .

**

Изменим направление оси на обратное (-).



Вместо V3 подставим V2. Вместо V2 подставим V1

; ;

С учетом матрицы A:

Видовые координаты объекта представляют собой аксонометрическую проекцию этого объекта на координатную плоскость . Координаты можно непосредственно использовать для формирования изображения на экране.

## 22. Перспективные преобразования (вывод матрицы преобразования). Схема пересчёта координат при переходе от мировых 3D-координат к экранным























Картинная плоскость – плоскость проецирования в видовой системе координат.

М(xe, ye) – точка на плоскости проектирования P.

– координаты точки схода лучей.

При такой расстановке координат

– z координата объекта в видовой системе координат.

M’(xe’,ye’) – изображение точки M в картинной плоскости без учета перспективы.

Из подобия треугольников и получаем пропорцию:

*;* таккак *, ;*

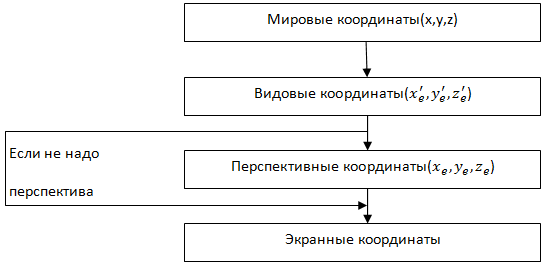
Аналогично их подобия треугольников и

Обозначим , тогда:

; ;

;

Т.е. преобразование координат при переходе от мировых к видовым включает следующие этапы:



## 23. Модели описания поверхностей

Аналитическая модель

Аналитической моделью будем называть описание поверхности математиче­скими формулами. В КГ можно использовать много разновидностей такого описания. Например, в виде функции двух аргументов *z =F(х, у).* Можно использовать уравнение *F (х, у, z)* = 0.

Зачастую используется параметрическая форма описания поверхности. За­пишем формулы для трехмерной декартовой системы координат *(х, у, z):*

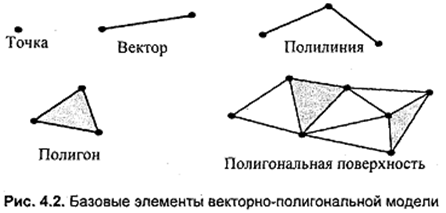


где s и t — параметры, которые изменяются в определенном диапазоне, функции *Fx*, *Fy* и *Fz* будут определять форму поверхности.

Преимущества параметрического описания — легко описывать поверхности, которые отвечают неоднозначным функциям, замкнутые поверхности.

Векторная полигональная модель

Для описания пространственных объектов здесь используются такие элементы: вершины; отрезки прямых (векторы); полигоны, полигональные поверхности(рис. 4.2).

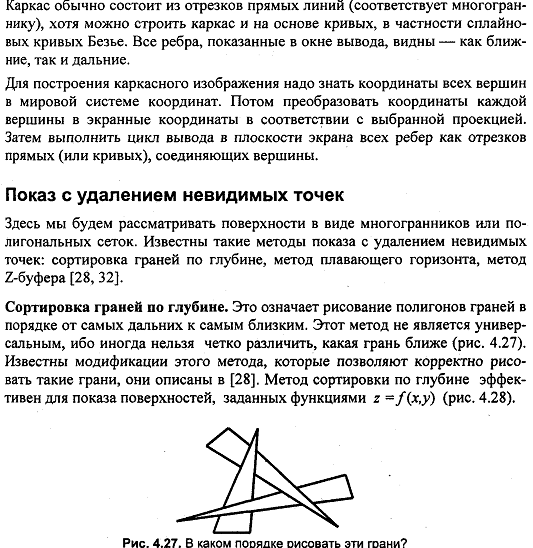


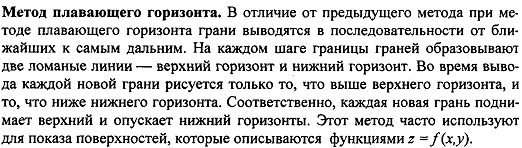
Элемент "вершина" (vertex) – главный элемент описания, все другие являются производными. При использовании трехмерной декартовой систем координаты вершин определяются как(xi, yi, zi).Каждый объект однозначно определяется координатами собственных вершин.

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

Векторную полигональную модель можно считать наиболее распространен­ной в современных системах трехмерной КГ.

## 24. Каркасная визуализация трёхмерных изображений. Принцип удаления невидимых граней для выпуклого многогранника. Реализация

****

****

## 25. Изображение поверхности z = f(x,y). Метод сортировки граней по глубине (алгоритм художника). Реализация

Поверхности определяется как множество точек, координаты которых удовлетворяют определённому виду уравнений:

F(x,\,y,\,z)=0\qquad (1)

Если функция F(x,\,y,\,z) непрерывна в некоторой точке и имеет в ней непрерывные частные производные, по крайней мере одна из которых не обращается в нуль, то в окрестности этой точки поверхность, заданная уравнением (1), будет правильной поверхностью.

Помимо указанного выше неявного способа задания поверхность может быть определена явно, если одну из переменных, например z, можно выразить через остальные:

z=f(x,y)\qquad (1')

Также существует параметрический способ задания.

Алгоритм художника

Алгоритм сортировки по глубине – поверхности объектов сортируются по удаленности от наблюдателя и заполняются соответствующими цветами, начиная с самой дальней.

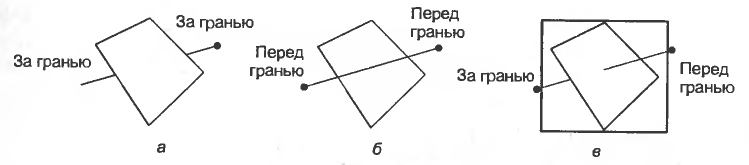
Алгоритм сортировки по глубине используется для удаления невидимых поверхностей. При применении алгоритма невидимых граней к множеству объектов удалено будет лишь около 50% невидимых линий. Нужен алгоритм, который удалял бы все невидимые линии независимо от количества объектов, их выпуклости и наличия криволинейных поверхностей.

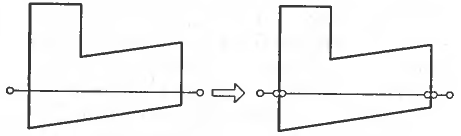
Один из таких алгоритмов действует следующим образом. Для каждого ребра (ребром объекта называется кривая пересечения соседних поверхностей, ограничивающих внутренний объем объекта) каждого объекта производится проверка, не закрыто ли оно гранями (гранями называются поверхности, ограничивающие объем объекта; площадь любой грани конечна, потому что все грани ограничиваются ребрами) каких-либо объектов. Закрытые части ребер последовательно исключаются до тех пор, пока не останется непроверенных поверхностей. Оставшиеся части ребер выводятся на экран. Реализация алгоритма включает несколько этапов.

1. Поверхности, направленные к наблюдателю, выделяются из всех остальных при помощи алгоритма невидимых граней. Выделенные поверхности сохраняются в массив FACE-TABLE. Грани, направленные от наблюдателя, учитывать не требуется, поскольку они сами по себе скрыты, а потому не скрывают ребра других граней. Для каждой грани сохраняется максимальное и минимальное значение Zv. Криволинейные поверхности разделяются по силуэтным линиям (как в алгоритме невидимых граней), а видимые части этих поверхностей также сохраняются в массиве FACE-TABLE. Вместе с плоскими гранями.
2. Ребра граней из массива FACE-TABLE выделяются из всех прочих ребер и собираются в отдельный список. Ребра других граней, не входящих в FACE-TABLE, можно не рассматривать, поскольку они невидимы. Затем для каждого ребра из списка производится проверка, не закрывается ли это ребро гранью из FACE-TABLE.
3. Скрытие ребра гранью можно обнаружить, сравнивая диапазоны значений Zv ребра и грани. Возможны три случая (рис. 45). В случае рис. 45 (а), все значения Zv ребра меньше минимального значения Zv грани, то есть, грань находится перед ребром. В случае рис. 45 (б) значения Zv ребра больше максимального значения Zv грани, то есть грань находится за ребром. В случае рис. 45 (в) диапазоны значений Zv грани и ребра перекрываются, то есть часть ребра находится за гранью, а другая часть – перед ней. Если ребро находится перед проверяемой гранью, из массива FACE-TABLE выбирается следующая грань и ребро сравнивается уже с ней. Если ребро оказывается за гранью, или проходит ее насквозь, приходится выполнять дополнительное действие.

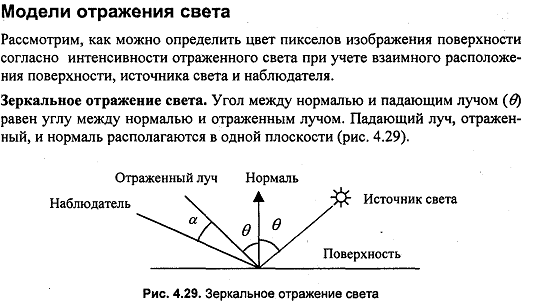
Ребро и грань проецируются на экран, после чего производится проверка перекрытия проекций. Если перекрытия нет, из этого следует, что ребро не закрывает проверяемую грань. Из массива FACE-TABLE выбирается следующая грань и проверяется согласно пункту 3. Если проекции перекрываются, ребро делится на две части по той точке, где она проходит сквозь проверяемую грань (рис. 46). Закрытая часть ребра отбрасывается, а видимые части добавляются в список. Затем пункт 3 повторяется для новых элементов списка. Исходное ребро удаляется из списка.

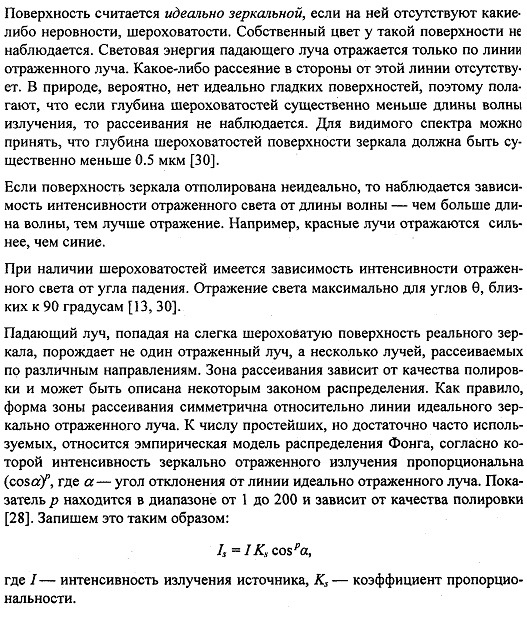
Ребра, прошедшие проверку со всеми гранями из FACE-TABLE, считаются видимыми и выводятся на экран.



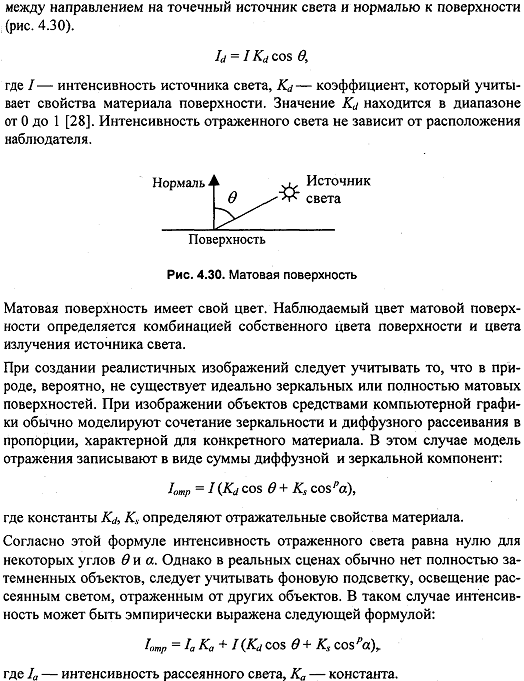


## 26. Закрашивание поверхностей. Модели отражения света

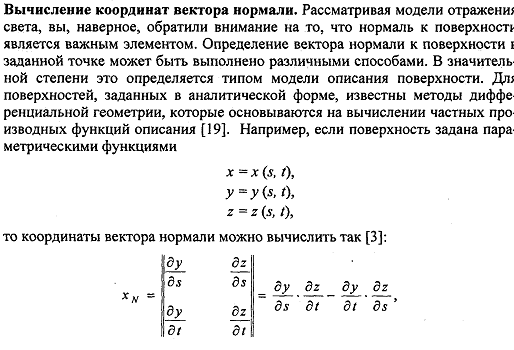
****

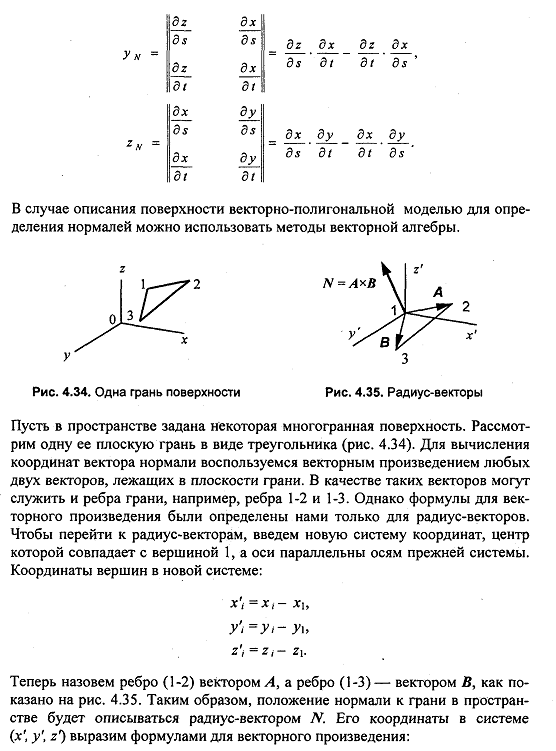
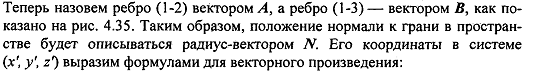
****

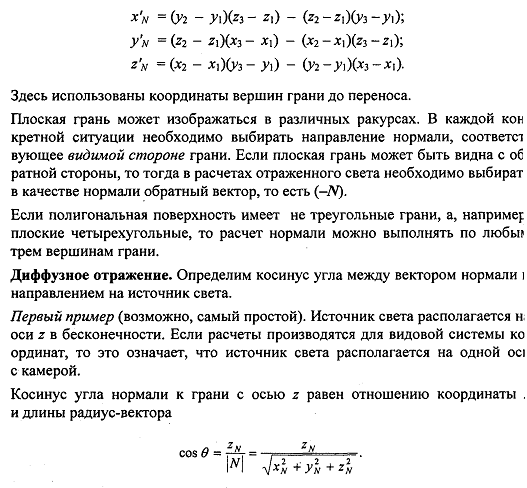
****

****

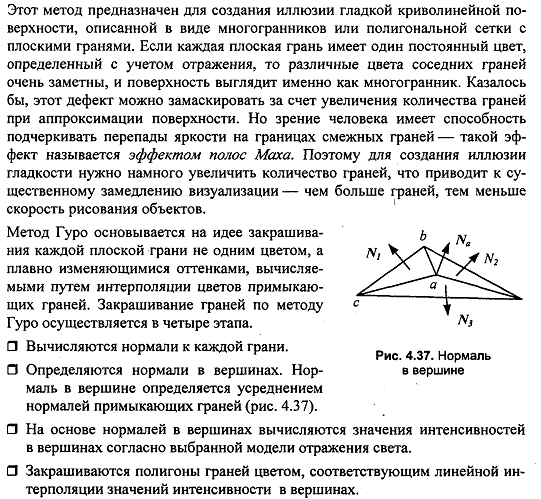
## 27. Закрашивание поверхностей. Вычисление нормалей и углов отражения. Реализация

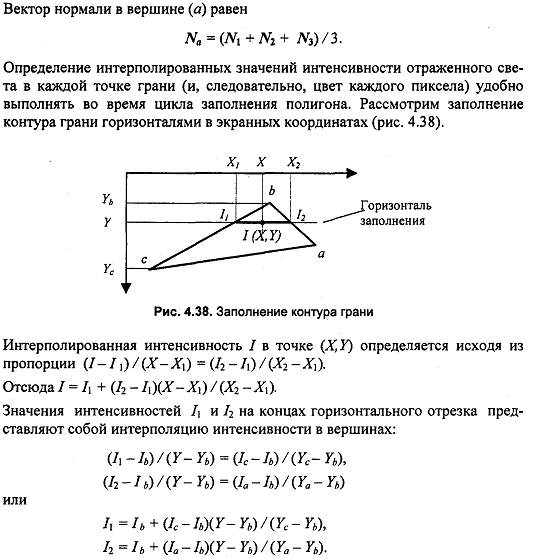
****

****  




## 28. Метод закрашивания Гуро

****

****