**Компьютерная геометрия и графика**

1. **Цели и задачи компьютерной графики [1,2,3].**

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. 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, Matcad) 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.

* Распознавание образов

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.

1. **Растровая и векторная визуализация изображений. Основные характеристики растровых изображений [1,2].**

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).

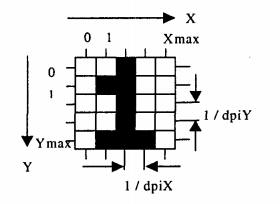
Недостаток растровых устройств – дискретность изображения. Недостатки векторных устройств – проблемы при сплошном заполнении фигур, мень­шее количество цветов, меньшая скорость (в сравнении с растровыми уст­ройствами).

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.

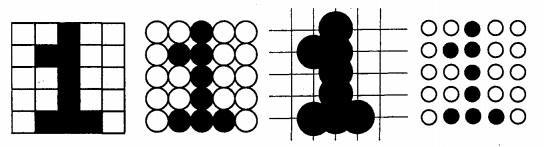
Геометрические характеристики растра

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.

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.

1. **Аддитивная цветовая модель RGB. Кодирование цвета. Палитра[1,2].**

Для того чтобы компьютер имел возможность работать с цветными изобра­жениями, необходимо представлять цвета в виде чисел – *кодировать цвета*. 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 под обрать 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.

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.

1. **Графические форматы. Работа с растровыми изображениями (форматы DDB, DIB, bmp – файлы) [1,2,3].**

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.

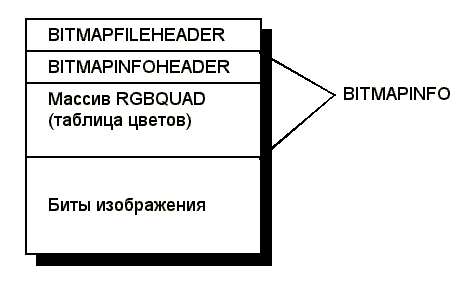
* Векторный формат наиболее удобен для хранения изображений, которые можно разложить на простые геометрические фигуры (например, чертежи или текст). Векторные файлы содержат математические описания элементов изображения. Наиболее распространенные векторные форматы: AutoCAD DXF и Microsoft SYLK.
* Растровый формат используется для хранения растровых данных. Файлы такого типа особенно хорошо подходят для хранения изображений реаль­ного мира, например оцифрованных фотографий. 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.

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. 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().

1. **Понятие об инкрементных алгоритмах растеризации. Растровое представление отрезка (вывод алгоритма). Реализация алгоритма (Листинг функции по лабораторной работе) [1]**

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 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.

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**

Пусть на растре заданы две точки и  с целочисленными координатами (рис. 4.1).

Запишем уравнение прямой, проходящей через эти точки



или

 (4.1)

где

,,,



















Рис. 4.1

Будем рассматривать случай для и .

Пусть точка  уже поставлена. Необходимо принять решение о том, какие координаты должна иметь точка:  или .

Запишем выражение для *y*-координаты математической прямой для :



Вычислим разности

,

,



.

Тогда



Выполним замену в





или



Полагая, что ,получим

, (4.2)

где

 (4.3)

 (4.4)

Так как , то



Вычитая (4.2) из (4.4), получим

.

Но, тогда .

При этом

.

Поэтому

.

Вычислим начальное значение .

Из (4.2) получаем

 (4.5)

Значение  соответствует точке .

Точка  по условию удовлетворяет уравнению прямой (4.1).

Отсюда

.

Подставляя полученное значение  в выражение (4.3)для , получим



.

Подставляя полученное выражение для в выражение (4.5) для , получим

.

|  |
| --- |
| . |
|  |
| . |

**Реализация в Matcad**

















т к т б о л ь ш е 0, т о х =х 0+1 и у =у 0+1



















































































1. **Понятие об инкрементных алгоритмах растеризации. Алгоритм построения окружности методом средней точки (вывод алгоритма). Реализация алгоритма. (Листинг функции по лабораторной работе) [1].**

**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 построения окружности методом средней точки***

Рассмотрим уравнение окружности радиуса с центром в начале координат

 (4.6)

Введем в рассмотрение функцию окружности

 (4.7)

Любая точка ,которая лежит на окружности удовлетворяет уравнению.

Если точка находится внутри круга, то функция окружности будет иметь отрицательное значение. Если точка лежит за пределами круга, значение функции окружности будет положительным. Подытоживая, можно сказать, что относительное положение любой точки с координатами определяется проверкой знака функции окружности точка:

 4.8)

Проверка выполняется на каждом этапе выборки для средних положений между пикселями вблизи заданной окружности. Таким образом, функция окружности–это параметр принятия решения в алгоритме средней точки, и для этой функции можно установить операции приращения, как это было сделано для алгоритма построения прямой линии.

На рис. 4.2 показана средняя точка между двумя возможными пикселами в точке выборки. Предположим, что мы только что поставили точку в пикселе с координатами . Теперь нужно определить, какой из двух пикселов ближе к заданной окружности – пиксел с координатами  или пиксел с координатами . Параметром принятия решения будет функция окружности (4.7), которая рассчитывается для средней точки между этими двумя пикселами:















Рис. 4.2

В случае, когда , полагаем (если средняя точка внутри окружности, то она (окружность) ближе к верхнему пикселу). Тогда







где



При этом



Тогда

, 

,  , для .

В случае, когда , делаем шаг вниз (если средняя точка вне окружности, то она(окружность) ближе к нижнему пикселу), полагая . Тогда







,

где



.

Для рассматриваемого случая, когда , , получаем



При этом. Тогда





Таким образом

, 

, 

для .

Рассмотрим алгоритмы вычисления и  для случая, когда , .

В этом случае

.

И выражение дляпринимает вид



Таким образом, если , то



,

если , то



.

Для удобства заменим в выражениях дляи  на , тогда для



При получаем



Начальные значения параметров и  определим из условия  и .

Тогда

.

Поскольку радиус  и приращение  целые числа, то значение  можно округлить и принять









Подводя итог изложенному, запишем в собранном виде ***алгоритм построения окружности методом средней точки:***

1. Установить начальные значение для .
2. Поставить начальные точки.
3. Выполнять в цикле пока 
4. если , то

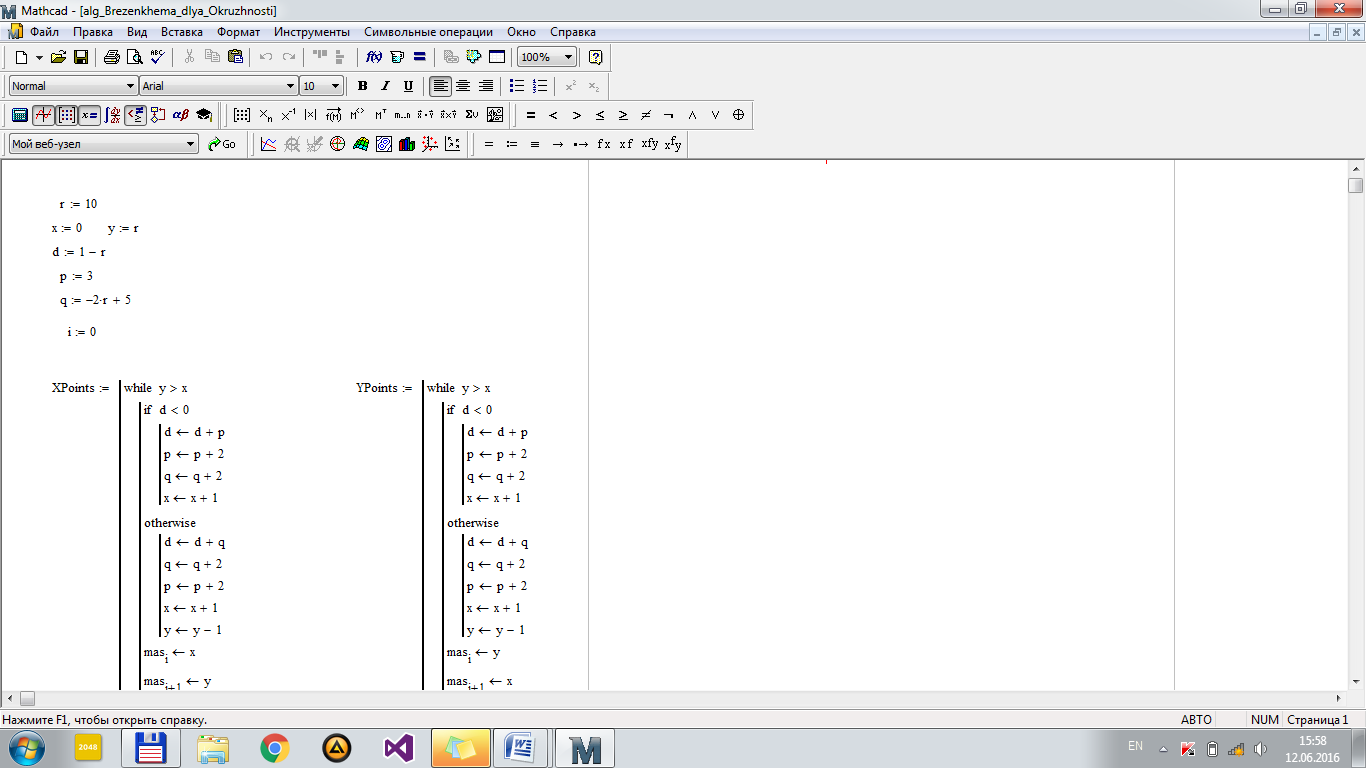


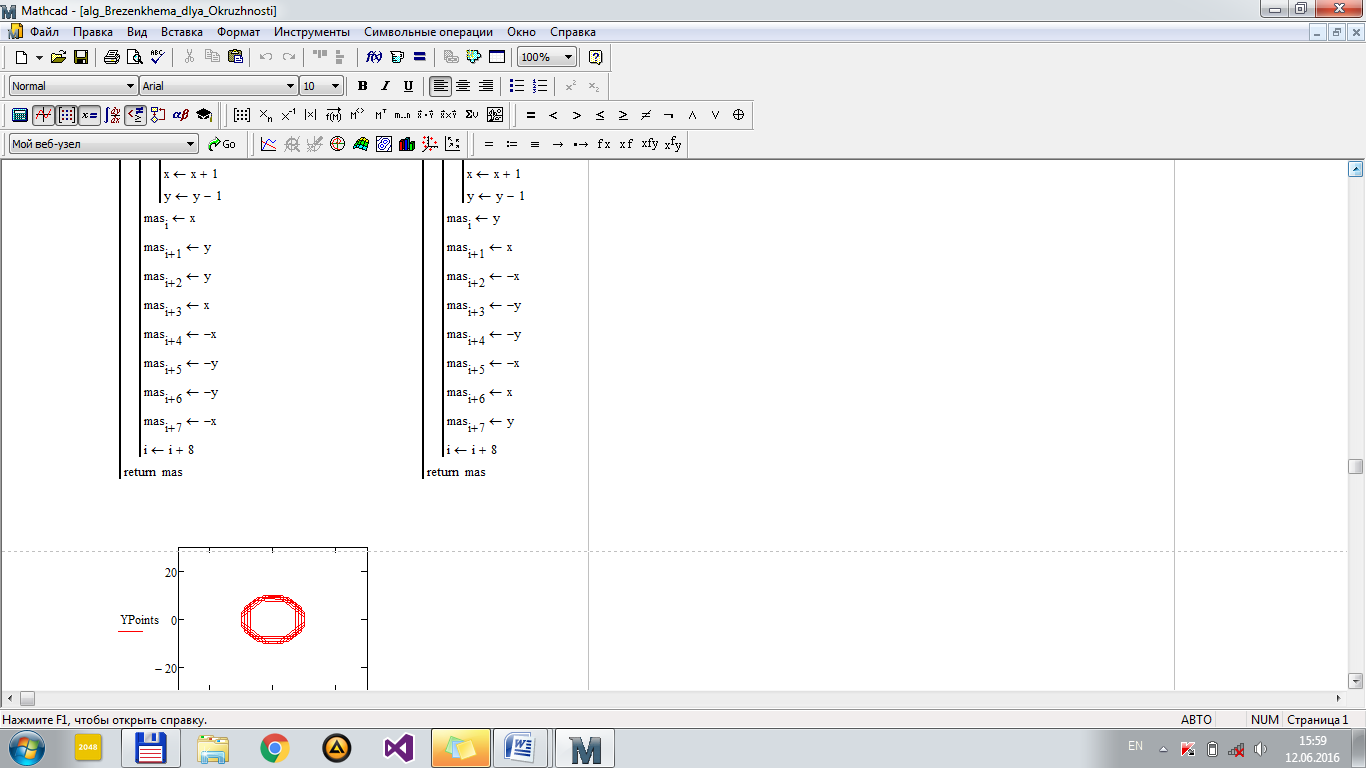
1. если , то



1. Поставить точки, соответствующие текущему шагу (рис. 4.3)

**Реализация в Matcad**





1. **Стиль линии (перо). Алгоритмы вывода линий [1,2]**

***Стиль линии. Перо***

Для описания различных по виду изображений на основе линий используют термин **стиль линий** или **перо**. Термин перо иногда делает более понятной суть алгоритма вывода линий для некоторых стилей – в особенности для толстых линий. Например, если для тонкой непрерывной линии перо соответствует одному пикселу, то для толстых линий перо можно представить себе как фигуру или отрезок линии, который скользит вдоль оси линии, оставляя за собой след (рис.4.4).

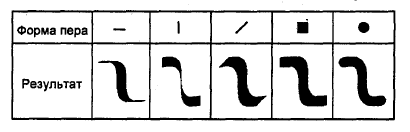


Рис. 4.4

***Алгоритмы вывода толстой линии***

Взяв за основу любой алгоритм вывода обычных тонких линий (например, алгоритм Брезенхэма), запишем его в следующем обобщенном виде:

*Вывод пиксела (х, у)*

Можно представить себе такой алгоритм, как цикл, в котором определяются координаты каждого пиксела. Этот алгоритм можно модифицировать для вывода толстой линии следующим образом:

*Вывод фигуры (или линии) пера с центром (х, у)*

Вместо вывода отдельного пиксела стоит вывод фигуры или линии, соответствующей перу – прямоугольник, круг, отрезок прямой.

Такой подход к разработке алгоритмов толстых линий имеет преимущества и недостатки. Преимущество – можно прямо использовать эффективные алгоритмы для вычисления координат точек линии оси, например, алгоритмы Брезенхэма. Недостаток – неэффективность для некоторых форм пера. Для перьев, которые соответствуют фигурам с заполнением, количество тактов работы алгоритма пропорционально квадрату толщины линии. При этом большинство пикселов многократно закрашивается в одних и тех же точках (рис. 4.5).

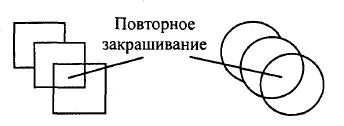


Рис. 4.5

Такие алгоритмы более эффективны для перьев в виде отрезков линий. В этом случае каждый пиксел рисуется только один раз. Но здесь важным является наклон изображаемой линии. Ширина пера зависит от наклона (рис. 4.6).

Очевидно, что горизонтальное перо не может рисовать толстую горизонтальную линию.

Для вывода толстых линий с помощью пера в качестве отрезка линии чаще всего используются отрезки горизонтальной или вертикальной линий, реже – диагональные отрезки под углом 45 градусов. Целесообразность такого способа определяется большой скоростью вывода горизонтальных и вертикальных отрезков прямой. Для того, чтобы достигнуть минимального количества тактов вывода, толстые линии, которые по наклону ближе к вертикальным, рисуют горизонтальным пером, а пологие линии – вертикальным пером.

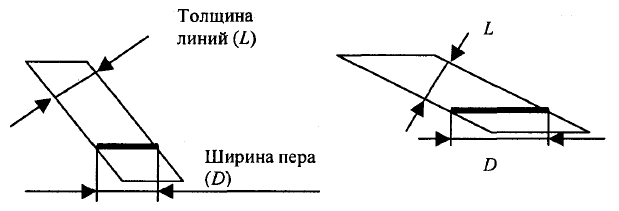


Рис. 4.6

***Алгоритмы вывода пунктирной линии***

Алгоритм для рисования тонкой пунктирной линии можно получить из алгоритма вывода тонкой непрерывной линии заменой процедуры вывода пиксела более сложной конструкцией:

*Вывод пиксела* (х, у)

*Проверка значения счетчика С :*

*Если С удовлетворяет некоторым условиям,*

*то вывод пиксела (х, у)*

*Значение С увеличивается на единицу*

При выводе по линии, которые состоят из отрезков прямых, необходимо предотвратить обнуление значения счетчика в начале каждого отрезка и обеспечить продолжение непрерывного приращения вдоль всей сложной линии. Иначе будут нестыковки пунктира. Использование переменной-счетчика затруднено при генерации пунктирных линий в алгоритмах, которые используют симметрию, например, при выводе круга или эллипса. В этом случае будут нестыковки пунктира на границах октантов или квадрантов.

1. **Стиль заполнения, кисть, текстура [1,2]**

При выводе фигур могут использоваться различные стили заполнения. Для обозначения стилей заполнения, отличных от сплошного стиля, используют такие понятия, как **кисть** и **текстура**. Их можно считать синонимами, однако понятие текстуры обычно используется применительно к трехмерным объектам, а кисть – для изображения двумерных объектов. Текстура – это стиль заполнения, закрашивание, которое имитирует сложную рельефную объемную поверхность, выполненную из какого-то материала.

Для описания алгоритмов заполнения фигур с определенным стилем можно использовать тот же способ, что и для описания алгоритмов рисования линий. Описание всех разновидностей подобных алгоритмов можно дать с помощью обобщенной схемы:

***Вывод пиксела заполнения цвета С с координатами (х, у)***

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



***Вывод пиксела заполнения *цветом**

Функция  будет определять стиль заполнения. Аргументами функции цвета являются координаты текущего пиксела заполнения. Однако в отдельных случаях эти аргументы не нужны. Например, если цвет С вычислять как случайное значение в определенных границах: , то можно создать иллюзию шершавой матовой поверхности (рис.4.7, а).

|  |  |
| --- | --- |
|  |  |
| а) | б) |
| Рис. 4.7 | |

Другой стиль заполнения – штриховой (рис. 4.7, б). Для него функцию цвета также можно записать в аналитической форме:

,

где – период, – толщина штрихов,– цвет штрихов, цвет фона.

Подобную функцию можно записать и для других типов штриховки. Аналитическая форма описания стиля заполнение позволяет достаточно просто изменять размеры штрихов при изменениях масштаба показа, например, для обеспечения режима WYSIWYG.

1. **Геометрические основы компьютерной графики. Системы координат (декартова, полярная, сферическая) и векторы. Скалярное и векторное произведение векторов [1,2,3].**

Системы координат и векторы

Рассмотрим различные способы представления векторов на плоскости и в пространстве. Пусть  некоторый вектор, заданный в декартовой системе координат  на плоскости (рис.7.1, *а*) и декартовой системе координат  в пространстве (рис. 7.1, *б*).

|  |  |
| --- | --- |
|  |  |
| *а*) | *б*) |

Рис. 7.1

В прямоугольной системе координат  направление осей задается тройкой перпендикулярных единичных векторов , , . Система координат называется **правой,** если при повороте от вектора  к вектору  на  направление вектора совпадает с поступательным движением винта с правой резьбой (рис. 7.2). Начальная точка векторов обозначается буквой *О*. Ниже представлены различные формы записи векторов на плоскости и в пространстве















Рис. 7.2

 (7.1)

 (7.2)

Другая форма записи векторов

 (7.3)

 (7.4)

Модуль (длина) вектора

 (7.5)

 (7.6)

Пусть заданы два вектора и в форме (7.4)

и,

Тогда

 (7.7)

 (7.8)

При этом справедливы равенства

 (7.9)

***Скалярное произведение векторов***

Скалярное произведение векторов  и определяется выражением

, (7.10)

где - угол между векторами  и 

В координатной форме

 (7.11)

или

, (7.12)

если вектора  и  представлены в форме (7.4).

Из (7.10) получаем

 (7.13)

Если , то угол между векторами  и  является острым , при– вектора и  ортогональны, а если , то угол между векторами является тупым .

Отметим некоторые свойства скалярного произведения.

Пусть ,,–некоторые числа. Тогда справедливы равенства:

 (7.14)

***Векторное произведение векторов***

Векторное произведение векторов  и  определяется выражением

, (7.15)

где  – угол между векторами  и . Направление вектора  перпендикулярно векторам  и и таково, что вектора , ,  в приведенной последовательности образуют правостороннюю тройку (рис. 7.2).

Отметим некоторые свойства векторного произведения.

Длина вектора равна

 (7.16)

Если, где  скаляр, то

, (7.17)

так как  (см. ниже).

Пусть  – некоторая константа, тогда справедливы соотношения

 (7.18)

Рассмотрим еще одну форму записи векторного произведения



. (7.19)

Полярная система координат

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Приведем в качестве справки соотношения, связывающие координаты точки , в декартовой и полярной системе координат (рис. 7.8).   |  |  | | --- | --- | |  | (7.59) | | Рис. 7.8 |  |   ***Сферическая система координат***  Ниже приведены соотношения, связывающие координаты точки , в декартовой и сферической системе координат (рис. 7.9).   |  |  | | --- | --- | |  | (7.60) | | Рис. 7.9 |  | |  |

1. **Геометрические основы компьютерной графики. Уравнение прямой, проходящей через две точки (на плоскости). Параметрическое уравнение прямой. Уравнение отрезка прямой. Задача о взаимном расположении двух отрезков [1,2,3].**

***Уравнение прямой на плоскости***

Пусть в декартовой системе координат на плоскости  заданы две точки

 и .

Тогда уравнение прямой, проходящей через эти точки имеет вид:

 (7.20)

Полагая в (7.20)

,

получаем параметрическое уравнение прямой на плоскости, проходящей через точки  и .

, (7.21)

Пусть.

Тогда (7.21) можно записать в виде

, (7.22)

или в более компактной форме

. (7.23)

Из (7.23) следует, что при , а при  получаем, что .

Таким образом, при точка  принадлежит отрезку .

Следовательно, при, уравнение (7.23) можно считать уравнением отрезка .

Рассмотрим задачу о взаимном расположении двух отрезков на плоскости.

***Условие задачи***. *На плоскости заданы два отрезка прямой:и *

, ,

, .

***Определить:*** *пересекаются или нет отрезки и .*

Запишем уравнения прямых и , определяемых отрезкамии  соответственно в параметрической форме

 (7.24)

 (7.25)

Предположим, что прямые пересекаются, и  - точка их пересечения.

В этой точке справедливо равенство

.

или

.

Отсюда





или

. (7.26)

Определитель системы (7.26) равен

 (7.27)

Если , то система уравнений (7.26) решения не имеет отрезки параллельны (или накладываются).

Если , то система уравнений (7.26) имеет единственное решение  и .

При этом если  и , то отрезки пересекаются, иначе пересекаются их продолжения, или один отрезок пересекается с продолжением другого.

Подставляя найденные значения  и  в (7.24) или в (7.25), можно определить координаты точки пересечения отрезков .

 (7.28)

или

. (7.29)

***Уравнение прямой в пространстве***

Пусть теперь две точки заданы в пространстве

 и  (7.30)

Тогда уравнение прямой, проходящей через эти точки, будет иметь вид:

. (7.31)

И в параметрической форме

 (7.32)

Уравнения (7.32), как и ранее, можно представить в компактной форме

. (7.33)

Рассмотрим задачу о пересечении плоскости отрезком прямой.

***Условие задачи.*** *В декартовой системе координат задана плоскость , определяемая уравнением*

 (7.34)

*и отрезок прямой , где точки  и  определяются по*  и 

***Определить:*** *пересекает отрезок  плоскость  (рис. 7.3) или нет (рис. 7.4)*.

|  |  |
| --- | --- |
|  |  |
| Рис. 7.3 | Рис. 7.4 |

Для решения поставленной задачи определим значение параметра , при котором прямая (7.32) и плоскость (7.34) имеют общую точку.

Подставляя выражения для координат (7.32) в уравнение плоскости (7.34) и вводя обозначения , , , получим

 (7.35)

Из (7.35) получаем

 (7.36)

для .

Тогда, если , то отрезок  пересекает плоскость . В противном случае плоскость  пересекает продолжение рассматриваемого отрезка.

Координаты точки пересечения определяются путем подстановки значения в (7.32)

 , (7.37)

или в компактной форме

. (7.38)

Рассмотрим случай, когда

. (7.39)

Известно, что в уравнении плоскости коэффициенты ,  и  являются координатами вектора , нормального к этой плоскости, т. е.. Рассмотрим вектор , построенный на отрезке . Тогда условие (7.39) означает, что

. (7.40)

Равенство нулю скалярного произведения (7.40) означает, что вектор перпендикулярен вектору нормали и, следовательно, параллелен плоскости . Если при этом также справедливо равенство , то вектор (отрезок ) лежит в плоскости .

Рассмотрим один частный случай рассмотренной задачи, когда плоскость (7.34) параллельна плоскости  и определяется уравнением  или

. (7.41)

В этом случае , ,  и . Подставляя эти значения в (7.36), получаем

 (7.42)

Подставив найденное значение в (37), находим

 (7.43)

1. **Построение кривых. Интерполяционный полином Лагранжа. Достоинства и недостатки. (Листинг функции для реализации полинома Лагранжа по лабораторной работе) [4].**

Современная научная компьютерная графика дает возможность проводить вычислительные эксперименты с наглядным представлением их результатов в виде кривых на плоскости и поверхностей в пространстве.

***Интерполяция функций***

**Интерполяция** – в вычислительной математике способ нахождения промежуточных значений величины по имеющемуся дискретному набору известных значений. На основании наборов значений, полученных экспериментальным путём или методом случайной выборки требуется построить функцию, на которую могли бы с высокой точностью попадать другие получаемые значения. Такая задача называется **аппроксимацией**. **Интерполяцией** называют такую разновидность аппроксимации, при которой кривая построенной функции проходит точно через имеющиеся точки данных.

***Интерполяционный полином Лагранжа***

Будем искать функцию  в виде полинома заданной степени  – **интерполяционного полинома**, т. е. положим что

,

где

 (14.1)

и в соответствии с определением интерполирующей функции

 (14.2)

Поскольку степень полинома известна, то поиск полинома сводится к нахождению набора его коэффициентов .

Французский математик **Лагранж** предложил способ построения интерполяционного полинома без предварительного вычисления коэффициентов , т. е. без решения системы уравнений (14.3).

Будем искать интерполяционный полином, который в данном случае обозначим через, в виде

(14.6)

Неизвестные коэффициенты  определим из условия .

Последовательно полагая , получим





Подставляя найденные значения коэффициентов  в выражение (14.6) для многочлена , получим

 (14.7)

Полученный таким образом полином называется интерполяционным полиномом Лагранжа.

Следует отметить, что полином Лагранжа  просто представляет собой другую форму записи рассмотренного ранее полинома , что следует из единственности решения задачи интерполяции.

Выражение для полинома Лагранжа  может быть легко преобразовано к виду  путем группировки коэффициентов при соответствующих степенях аргумента .

Примеры.

Пусть имеем две точки  и , что соответствует значению . Тогда

.

Обозначая  и , получаем

.

Пусть имеем три точки ,  и , что соответствует значению . Тогда



***Достоинства полинома Лагранжа:***

* график интерполяционного многочлена Лагранжа проходит через каждую точку массива;
* конструируемая функция легко описывается (число подлежащих определению коэффициентов интерполяционного многочлена Лагранжа на сетке равно);
* построенная функция имеет непрерывные производные любого порядка;
* заданным массивом интерполяционный многочлен определен однозначно.

***Недостатки полинома Лагранжа:***

* степень интерполяционного многочлена Лагранжа зависит от числа узлов сетки, и чем больше это число, тем выше степень интерполяционного многочлена и, значит, тем больше требуется вычислений;
* изменение хотя бы одной точки в массиве требует полного пере­счета коэффициентов интерполяционного многочлена Лагранжа.

1. **Построение кривых. Интерполяционный кубический сплайн. Реализация**

**Сплайн**(англ. spline – планка, рейка) – функция, область определения которой разбита на конечное число отрезков, на каждом из которых сплайн совпадает с некоторым алгебраическим полиномом. Сплайны имеют многочисленные применения как в математической теории, так и в разнообразных вычислительных приложениях. В частности, сплайны двух переменных интенсивно используются для задания поверхностей в различных системах компьютерного моделирования.

***Интерполяционный кубический сплайн***

Рассмотрим наиболее известный и широко применяемый интерполяционный сплайн степени 3 дефекта 1. При этом будем исходить из предположения, что узлы сплайна

 (14.8)

одновременно служат узлами интерполяции, т.е. в них известны значения функции**, .

***Определение****.* Кубическим сплайном дефекта 1,интерполирующим на отрезке  данную функцию , называется функция

, (14.9)

где

 (14.10)

удовлетворяет совокупности условий:

 – (14.11)

условие интерполяции в узлах сплайна,

– (14.12)

двойная непрерывная дифференцируемость,

 – (14.13)

краевые (граничные) условия.

Заметим, что граничные условия вида (14.13) называются естественными граничными условиями.

Определенный таким образом сплайн называют еще **есте­ственными** или **чертежным сплайном** и связано это со следующим обстоятельством. Желая провести плавную линию через заданные точки плоскости, чертежники фиксировали в этих точках гибкую упругую рейку, тогда под влиянием упругих сил она принимала нужную форму, обеспечивающую минимум потенциальной энергии.

Для построения по данной функции интерполирующего ее сплайна (14.9) нужно найти  его коэффициентов 

Имеем:

из условий интерполяции (14.11) для функции

,  при  (14.14)

из условий гладкой стыковки звеньев сплайна (14.12)

, при  (14.15)

из краевых условий (14.13)

, (14.16)

Подставляя сюда выражения (14.9) для функций ,

 ,

их производных

 (14.17)

и

 (14.18)

через коэффициенты  при указанных значениях и, полагая для краткости

, (14.19)

Рассмотрим случай равноотстоящих узлов.

Тогда  и расчетные формулы для коэффициентов полинома [7] принимают вид:

, 

Для  вычисляем

, .

Полагаем  и для вычисляем

.

Полагаем  и с учетом получаем

, .

В результате при значение  можно заменить значением

.

с найденными значениями коэффициентов .

***Достоинства кубической сплайн-интерполяции:***

–график построенной функции проходит через каждую точку массива;

–конструируемая функция сравнительно легко описывается (число подлежащих определению коэффициентов равно );

–заданным массивом построенная функция определена однозначно;

–степень многочленов не зависит от числа узлов сетки и, следовательно, не изменяется при его увеличении;

–построенная функция имеет непрерывные первые и вторые производные;

–построенная функция обладает хорошими аппроксимационными свойствами.

К недостаткам кубических сплайнов относится то, что они склонны осциллировать в окрестностях точки, существенно отличающейся от своих соседей.

1. **Построение кривых. Геометрические сплайны. Кривая Безье. Геометрический алгоритм построения кривой Безье. (Листинг функции для реализации геометрического алгоритма построения кривой Безье по лабораторной работе) [1,2].**

***Геометрические сплайны***

Во многих задачах требование того, чтобы конструируемая кривая однозначно проектировалась соответственно на прямую, является слишком жестким. В качестве нового способа задания кривых удобно использовать параметрический способ.

Формулировка задачи: по заданному множеству вершинс учетом их нумерации построить гладкую кривую, которая, плавно изменяясь, последовательно проходила бы вблизи этих вершин и удовлетворяла некоторым дополнительным условиям. Эти условий могут иметь различный характер. Например, можно потребовать, чтобы искомая кривая проходила через все заданные вершины или, проходя через заданные вершины, касалась заданных направлений, являлась замкнутой или имела заданную регулярность и т. п.

При отыскании подходящего решения задачи приближения важ­ную роль играет ломаная, звенья которой соединяют соседние вершины заданного набора. Эту ломаную называют **контрольной** или **опорной**, а ее вершины – контрольными или опорными.

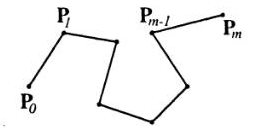


Рис. 14.2

Во многих случаях она довольно точно показывает, как будет проходить искомая кривая, что особенно полезно при решении задачи сглаживания. Каждая вершина заданного массива является либо **внутренней** либо **граничной**(концевой). В массиве Р вершины Р1 …, Pm-1 внутренние,а вершины Р0 и Рm– граничные (концевые).

Никаких ограничений на множество вершин не накладывается - они могут быть заданы как на плоскости, так и в пространстве, их взаимное расположение может быть совершенно произвольным, некоторые из вершин могут совпадать и т. д. Поэтому описание нужной кривой ищут в следующем виде:

 (14.20)

где– некоторые функциональные коэффициенты, подлежащие определению.

Если количество вершин в заданном множестве Р достаточно ве­лико, то найти универсальные функциональные коэффициенты, как правило, довольно затруднительно. Если универсальные коэффициенты  все же найдены, то часто оказывается, что они наряду с нужными свойствами обладают и такими, которые не всегда удовлет­ворительно согласуются с ожидаемым поведением соответствующей кривой (например, кривая, описываемая уравнением (14.20) с этими коэффициентами, может осциллировать или заметно отклоняться от заданного множества).

Для успешного решения поставленной задачи приближения, весьма удобно привлечь кривые, составленные из элементарных фраг­ментов. В случае, когда эти элементарные фрагменты строятся по еди­ной сравнительно простой схеме, такие составные кривые принято называть сплайновыми кривыми.

Параметрические уравнения каждого элементарного фрагмента ищутся в виде (14.20) с той лишь разницей, что всякий раз привлекается только часть заданных вершин множества Р. Для описания элементарных кривых и вычисления их геометрических характеристик (информация о которых необходима при состыковке) в качестве функциональных коэффициентов обычно используются многочлены невысоких степеней 2-й или 3-й, в первую очередь потому, что они сравнительно просто вычисляются. Наибольшее распространение получили методы конструирования составных кривых, в которых используются кубические многочлены.

***Кривая Безье***

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

Кривые Безье описываются в параметрической форме[2,5]:

 (14.21)

Значение *t* выступает как параметр, которому отвечают координаты отдельной точки линии. Параметрическая форма описания может быть более удобной для некоторых кривых, чем задание в виде функции. Это потому, что функция  может быть намного сложнее, чем и, кроме того, может быть неоднозначной.

Многочлены Безье для  иимеют такой вид:

, (14.22)

где – сочетание по , и – координаты точек-ориентиров. Значение  можно рассматривать и как степень полинома, и как значение, которое на единицу меньше количества точек-ориентиров.

Рассмотрим кривые Безье, классифицируя их по значениям .

***m*=1** (по двум точкам).

Кривая вырождается в отрезок прямой линии, определяемой концевыми точками (рис.14.3)

 , (14.23)

где.

|  |  |
| --- | --- |
|  |  |
| Рис. 14.3 | Рис. 14.4 |

***m*=2** (по трем точкам, рис. 14.3).

 (14.24)

***m*=3** (по четырем точкам, рис. 14.4 *а*,*б*).

(14.25)

|  |  |
| --- | --- |
|  |  |
| *А* | *б* |
|  |  |
| *в* | *г* |
| Рис. 14.4 | |

***Геометрический алгоритм для кривой Безье***

Этот алгоритм позволяет вычислить координаты  точки кривой Безье позначению параметра .

**Суть алгоритма заключается в следующем:**

* каждая сторона контура многоугольника, проходящего по точкам-ориентирам, делится пропорционально значению ;
* точки деления соединяются отрезками прямых и образуют новый многоугольник. Количество узлов нового контура на единицу меньше, чем количество узлов предыдущего контура;
* стороны нового контура снова делятся пропорционально значению . И так далее.

Это продолжается до тех пор, пока не будет получена единственная точка деления. Эта точка и будет точкой кривой Безье (рис. 14.5*а*, *б*).

|  |  |
| --- | --- |
|  |  |
| *а* | *б* |
| Рис. 14.5 | |

Приведем запись геометрического алгоритма на языке C++:

for(i=0; i<=m; i++)

R[i]=P[i]; // Формируем вспомогательный массив R

for(j=m; j>0; j--)

for(i=0; i<j; i++)

R[i]=R[i]+t\*(R[i+1]-R[i]);

Результат работы алгоритма – координаты одной точки кривой Безье – записываются в R[0].

1. **Мировые и экранные координаты. Алгоритм пересчета мировых 2D-координат в оконные. Вывод и реализация (Листинг функции SpaceToWindow)**

При отображении пространственных объектов на экране или на листе бумаги с помощью принтера необходимо знать координаты объектов. Мы рассмотрим две системы координат. Первая **–мировая система координат(МСК)**, которые описывают истинное положение объектов в пространстве с заданной точностью. Другая **– оконные координаты** или **система координат устройства изображения**, в котором осуществляется вывод изображения объектов в заданной проекции.

Мировые координаты объектов являются трехмерными. Положение объекта может быть описано, например, в прямоугольной или сферической системе координат. Где располагается центр системы координат и каковы единицы измерения вдоль каждой оси, не очень важно. Важно то, что для отображения должны быть известны какие-то числовые значения координат отображаемых объектов.

Первоначально рассмотрим работу с плоскими изображениями, для описания которых используются двухмерные мировые координаты.

***Рассмотрим задачу получения формул для пересчета мировых координат объекта в оконные.***

Пусть в мировом пространстве задана плоскость . Свяжем с этой плоскостью систему координат и выделим на ней прямоугольную область (рис. 5.1).

Рассмотрим оконную систему координат  с началом в левом верхнем углу окна и выделим в ней некоторую прямоугольную область (рис. 5.2).

Пусть  – некоторая точка в мировой системе координат, а  ее образ в оконной системе координат.

Из приведенных рисунков можно получить пропорции

 (5.1)













0









P

Рис. 5.1



0



















Рис. 5.2

 (5.2)

Введем обозначения:

|  |  |
| --- | --- |
|  | – ширина области отображения в оконных координатах, |
|  | – ширина области отображения в мировых координатах, |
|  | – высота области отображения в оконных координатах, |
|  | – высота области отображения в мировых координатах. |

Тогда выражение (5.2) можно переписать в виде

 (5.3)

или

, (5.4)

где , .

Представим соотношения (5.4) в матричном виде

 (5.5)

или

 (5.6)

где

, , .

Перепишем пару рассматриваемых равенств в виде, удобном для дальнейшего использования.

 (5.7)

Представим (5.7) также и в матричном виде

 , (5.8)

или

, (5.9)

где

, , 

**Листинг функции SpaceToWindow**

CMatrix SpaceToWindow(CRectD &RS, CRect &RW) //пересчет из миров в окон

{

// Возвращает матрицу пересчета координат из мировых в оконные

// rs - область в мировых координатах - double

// rw - область в оконных координатах - int

CMatrix M(3, 3);

double kx = (double)RW.Width() / (RS.right - RS.left); // Масштаб по x

double ky = (double)RW.Height() / (RS.top - RS.bottom); //Масштаб по y

//Заполнение матрицы пересчёта

M(0, 0) = kx;

M(0, 1) = 0;

M(0, 2) = (double)RW.left - kx\*RS.left;

M(1, 0) = 0;

M(1, 1) = -ky;

M(1, 2) = (double)RW.bottom + ky\*RS.bottom;

M(2, 0) = 0;

M(2, 1) = 0;

M(2, 2) = 1;

return M;

}

1. **Физическая и логическая системы координат. Режимы отображения (без настройки параметров). Алгоритм преобразование координат в GDI и его связь с алгоритмом пересчета мировых координат в оконные[1]**

**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. 6.1 показана физическая система координат для экрана видеомонитора.

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.





Рис.6.1

***Логическая система координат***

Приложения Windows могут использовать одну из нескольких логических координат, устанавливая соответствующий режим отображения в контексте отображения. При этом 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 (рис. 6.2).







Рис.6.2

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

int dc.SetMapMode(int nMapMode) // MFC

Параметр **nMapMode** может принимать одно из следующих значений, представленных в табл.6.1.

Таблица 6.1.

**Значения параметра nMapMode**

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

В любой момент времени приложение может определить номер режима отображения, выбранный в контекст отображения, используя функцию **GetMapMode**:

intdc.GetMapMode()// MFC

***Преобразование координат***

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

, (6.1)

где

 – логические координаты по оси X и Y соответственно,

– физические (экранные) координаты по оси  и  соответственно,

 – определяют расположение начала **логической** системы координат, по умолчанию ,

 – определяют расположение начала **физической** системы координат, по умолчанию ,

 – задают масштаб, который используется в процессе преобразования координат по оси ,

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

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.

Таким образом, логические координаты преобразуются в физические координаты.

1. **Режимы отображения и настройка их параметров (функции класса CDC MFC). Функция SetMyMode, назначение, параметры и реализация (листинг) [1].**

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;

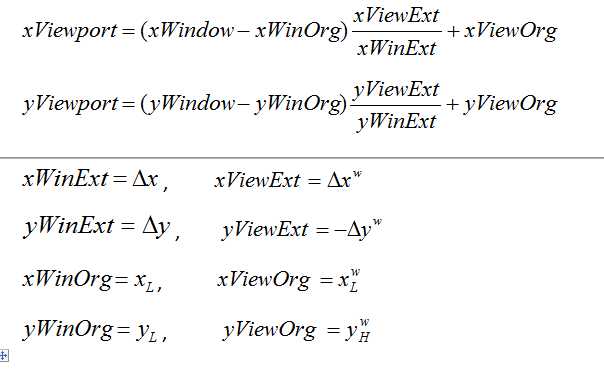
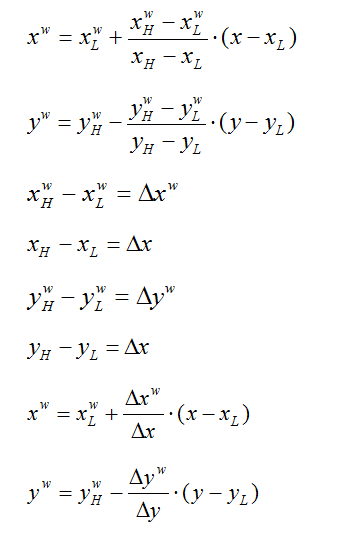
LONG cy;

} 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);  } |

1. **Аффинные преобразования на плоскости. Преобразования системы координат: смещение, растяжение-сжатие, поворот.[1,2]**

Пусть на плоскости задана система координат (СК) и точка  принадлежащая некоторому объекту. Система координат трансформируется в СК  путем ряда последовательных смещений и поворотов относительно своего исходного состояния. При этом точка (объект) остается **неподвижной**. Необходимо определить координаты точки (объекта) в системе координат  (рис. 8.1, *а*,*б*).

|  |  |
| --- | --- |
|  |  |
| *а*) | *б*) |
| Рис. 8.1 | |

В общем случае преобразование координат мочки при переходе от системы координат к системе координат определяется системой линейных уравнений:

 , (8.1)

где .

Выражение (8.1) представляет собой аффинное преобразование координат при переходе от системы координат к системе координат .

Обратный переход от СК к СК определяется как

, (8.2)

Аффинное преобразование (8.1) удобно представить в матричном виде:

 (8.3)

В компьютерной графике принято использовать ***однородные*** координаты, которые вводятся следующим образом. Точке  ставится в соответствие точка , а точке  точка .

Тогда переход от системы координат к системе координат в матричном виде можно записать как

 (8.4)

или

, (8.5)

где, ,

 (8.6)

Обратное преобразование

, (8.7)

где

. (8.8)

Рассмотрим теперь различные виды аффинных преобразований на плоскости и соответствующие им матрицы  и .

**Параллельный сдвиг системы координат** (рис. 8.2)

, (8.9)

 (8.10)

или

, (8.11)

|  |  |
| --- | --- |
|  |  |
| Рис. 8.2 | Рис. 8.3 |

где

. (8.12)

Здесь и далее индекс «» означает, что преобразованию подвергается система координат.

**Обратное преобразование**

Обратное преобразование соответствует перемещению системы координат в противоположном направлении. Соответствующие выражения для преобразования координат могут быть получены из (8.9) – (8.12) путем соответствующей замены в них → и  → .

Матрица преобразования будет иметь вид

 (8.13)

**Растяжение – сжатие системы координат** (рис. 8.3)

 (8.14)

 (8.15)

или

, (8.16)

где

 (8.17)

**Обратное преобразование**

Соответствующие выражения для преобразования координат могут быть получены из (8.14) – (8.17) путем соответствующей замены в них → и  → .

Приведем выражение для матрицы преобразования.

, (8.18)

**Поворот системы координат на угол** (рис. 8.4а, б)

|  |  |
| --- | --- |
|  |  |
| *а* | *Б* |
| Рис. 8.4 | |

Координаты точки  в системе координат :

 (8.19)

В системе координат :



С учетом (8.19), получаем:

 (8.20)

или

 (8.21)

, (8.22)

где

. (8.23)

**Обратное преобразование**

Обратное преобразование соответствует повороту системы координат  на угол–. Соответствующие выражения для преобразования координат могут быть получены из (8.19) – (8.23) путем замены в них → и  →–.

Матрица преобразования будет иметь вид

 (8.24)

1. **Аффинные преобразования на плоскости. Преобразования объектов: смещение, растяжение-сжатие, поворот [1,2].**

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

Пусть в системе координат  некоторая точка перемещается из положения  в положение (рис. 8.4,а,б) или обратно из положения  в положение .

|  |  |
| --- | --- |
| *а* | *б* |
| Рис. 8.5 | |

Тогда старые координаты точкии новые  связаны соотношениями (8.1) – (8.8). Подчеркнем лишь, что в данном случае выражения (8.1) – (8.8) описывают взаимосвязь между старыми и новыми координатами точки при изменении ее положения **в одной и той же** системе координат.

Рассмотрим частные случаи аффинных преобразований объектов.

**Сдвиг объекта** (рис. 8.6)

|  |  |
| --- | --- |
|  |  |
| Рис. 8.6 | Рис. 8.7 |

Из рис. 8.6 получаем

, (8.25)

 (8.26)

или

, (8.27)

где

. (8.28)

Здесь и далее индекс «» означает, что преобразованию подвергаются координаты объекта.

**Обратное преобразование**

Обратное преобразование соответствует перемещению объекта в противоположном направлении. Соответствующие выражения для преобразования координат могут быть получены из (8.25) – (8.28) путем соответствующей замены в них → и  → .

Матрица преобразования будет иметь вид

 (8.29)

**Растяжение – сжатие объекта**

 (8.30)

 (8.31)

или

, (8.32)

где

 (8.33)

**Обратное преобразование**

Соответствующие выражения для преобразования координат могут быть получены из (8.30) – (8.33) путем соответствующей замены в них → и  → .

Приведем выражение для матрицы преобразования.

, (8.34)

**Поворот объекта вокруг центра координат** (рис. 8. 7)

Координаты точки  в системе координат (рис. 8.7):

 (8.35)

Координаты точки  в системе координат :



С учетом (8.35), получаем:

 (8.36)

или

 (8.37)

, (8.38)

где

 (8.39)

**Обратное преобразование**

Обратное преобразование соответствует повороту объекта  на угол–. Соответствующие выражения для преобразования координат могут быть получены из (8.35) – (8.39) путем замены в них → и  →–.

Приведем выражение для матрицы преобразования.

 (8.40)

**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 (8.12) и (8.28) для сдвига

**,** (8.41)

выражений (8.18) и (8.33) для растяжения – сжатия

 , (8.42)

выражений (8.23) и (8.39) для поворота

 (8.43)

В Приложении 1 приведен пример приложения Windows, где используются аффинные преобразования на плоскости.

**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***

1. **Аффинные преобразования в пространстве. Преобразования системы координат: смещение, растяжение-сжатие, повороты вокруг осей координат [1,2].**

Пусть в пространстве задана система координат (СК)  и точка принадлежащая некоторому объекту. Система координат трансформируется в СК  путем ряда последовательных смещений и поворотов относительно своего исходного состояния. При этом точка (объект) остается **неподвижной**. Необходимо определить координаты точки  (объекта) в системе координат  (рис. 9.1 а, б).

|  |  |
| --- | --- |
|  |  |
| *А* | *б* |
| Рис. 9.1 | |

В общем случае преобразование координат мочки при переходе от системы координат к системе координат определяется системой линейных уравнений:

, (9.1)

где .

Выражение (9.1) представляет собой аффинное преобразование координат при переходе от системы координат к системе координат .

Обратный переход от СК к СК определяется как

, (9.2)

Аффинные преобразования (9.1) удобно представить вматричном виде:

 (9.3)

Как и ранее, введем в рассмотрение однородные координаты, когда точке ставится в соответствие точка , а точке  – точка .

Тогда переход от системы координат к системе координат в матричном виде можно записать как

, (9.4)

или

, (9.5)

где

, ,

. (9.6)

Обратное преобразование имеет вид

, (9.7)

где

. (9.8)

Рассмотрим теперь различные виды аффинных преобразований в пространстве и соответствующие им матрицы  и .

**Параллельный сдвиг системы координат** (рис. 9.2)

, (9.9)

|  |  |
| --- | --- |
|  |  |
| Рис. 9.2 | Рис. 9.3 |

, (9.10)

или

, (9.11)

где

. (9.12)

Здесь и далее индекс «» означает, преобразованию подвергается система координат.

**Обратное преобразование**

Обратное преобразование соответствует перемещению системы в противоположном направлении. Соответствующие выражения для преобразования координат могут быть получены из (9.10) – (9.13) путем соответствующей замены в них → и  → .

Матрица преобразования будет иметь вид

(9.13)

**Растяжение – сжатие системы координат** (рис. 9.3)

 (9.14)

, (9.15)

или

, (9.16)

где

. (9.17)

**Обратное преобразование**

Соответствующие выражения для преобразования координат могут быть получены из (9.15) – (9.18) путем соответствующей замены в них → и  → .

Приведем выражение для матрицы преобразования.

 (9.18)

**Поворот системы координат вокруг оси  на угол ** (рис. 9.4)

 (9.19)

 (9.20)

или

, (9.21)

|  |  |
| --- | --- |
|  |  |
| Рис. 9.4 | Рис. 9.5 |

где

. (9.22)

**Обратное преобразование**

Обратное преобразование соответствует повороту системы координат вокруг оси в противоположном направлении. Соответствующие выражения для преобразования координат могут быть получены из (9.20) – (9.23) путем замены в них → и  → .

Приведем выражение для матрицы преобразования.

 (9.23)

**Поворот системы координат вокруг оси  на угол ** (рис.9. 5)

 (9.24)

 (9.25)

или

, (9.26)

где

 (9.27)

**Обратное преобразование**

Обратное преобразование соответствует повороту системы координат вокруг оси в противоположном направлении. Соответствующие выражения для преобразования координат могут быть получены из (9.25) – (9.28) путем замены в них → и  → .

Приведем выражение для матрицы преобразования.

 (9.28)

**Поворот системы координат вокруг оси на угол ** (рис. 9.6)

 (9.29)

, (9.30)

или

, (9.31)

где

|  |
| --- |
|  |
| Рис. 9.6 |

 (9.32)

**Обратное преобразование**

Обратное преобразование соответствует повороту системы координат вокруг оси в противоположном направлении. Соответствующие выражения для преобразования координат могут быть получены из (9.30) – (9.33) путем замены в них → и  → .

Матрица преобразования для этого случая будет иметь вид

 (9.33)

1. **Аффинные преобразования в пространстве. Преобразования объектов: смещение, растяжение-сжатие, повороты вокруг осей координат [1,2].**

Как и ранее под преобразованием объекта будем понимать изменение координат точек, принадлежащих этому объекту при изменении его положения в некоторой системе координат.

Пусть в пространстве задана система координат  и точка принадлежащая некоторому объекту. Объект перемещается из одной точки пространства в другуюпутем ряда последовательных смещений и поворотов относительно своего исходного положения.При этом система координат остается **неподвижной**. Необходимо определить новые координаты точки  (объекта) в СК В общем случае старые координаты точкии новые  связаны соотношениями (9.1) – (9.9). Подчеркнём лишь, что в данном случае выражения (9.1) – (9.9) описывают взаимосвязь между старыми и новыми координатами точки при изменении ее положения **в одной и той же** системе координат.

Рассмотрим частные случаи аффинных преобразований координат объектов.

**Смещение объекта вдоль координатных осей** (рис.9.7)

, (9.34)

|  |  |
| --- | --- |
|  |  |
| Рис. 9.7 | Рис. 9.8 |

 (9.35)

или

, (9.36)

где

 (9.37)

Здесь и далее индекс «»означает, что преобразованию подвергаются координаты объекта.

**Обратное преобразование**

Обратное преобразование соответствует перемещению объекта в противоположном направлении. Соответствующие выражения для преобразования координат могут быть получены из (9.35) – (9.38) путем соответствующей замены в них → и  → .

Приведем выражение для матрицы преобразования.

 (9.38)

**Растяжение – сжатие объекта**

, (9.39)

, (9.40)

или

, (9.41)

где

 (9.42)

**Обратное преобразование.**

Соответствующие выражения для преобразования координат могут быть получены из (9.40) – (9.43) путем соответствующей замены в них → и  → .

Приведем выражение для матрицы преобразования.

(9.43)

**Поворот объекта вокруг оси  на угол **(рис. 9.8).

 (9.44)

, (9.45)

или

, (9.46)

где

 (9.47)

**Обратное преобразование**

Обратное преобразование соответствует повороту объекта вокруг оси  в противоположном направлении. Соответствующие выражения для преобразования координат могут быть получены из (9.45) – (9.48) путем соответствующей замены в них → и  → .

Приведем выражение для матрицы преобразования.

 (9.48)

**Поворот объекта вокруг оси  на угол **(рис. 9.9).

 (9.49)

|  |  |
| --- | --- |
|  |  |
| Рис. 9.9 | Рис. 9.10 |

 (9.50)

или

, (9.51)

где

. (9.52)

**Обратное преобразование**

Соответствующие выражения для преобразования координат могут быть получены из (9.50) – (9.53) путем соответствующей замены в них → и  → .

Приведем выражение для матрицы преобразования.

 (9.53)

**Поворот объекта вокруг оси  на угол **(рис. 9.10)

 (9.54)

 (9.55)

или

, (9.56)

где

 (9.57)

**Обратное преобразование**

Соответствующие выражения для преобразования координат могут быть получены из (9.55) – (9.58) путем соответствующей замены в них → и  → .

Приведем выражение для матрицы преобразования.

 (9.58)

***Связь преобразований объектов с преобразованиями координат***

Как было отмечено при рассмотрении аффинных преобразований на плоскости, движение объектов можно рассматривать как движение в обратном направлении соответствующей системы координат.

Запишем соответствующие соотношения, связывающие матрицы аффинных преобразований для перемещений объектов и системы координат, основываясь на сравнении выражений (9.13) и (9.38) для сдвига

, (9.59)

выражений (9.18) и (9.43) для растяжения – сжатия

, (9.60)

выражений (9.23), (9.28), (9.33) и (9.48), (9.53), (9.58) для поворотов

 (9.61)

 (9.62)

 (9.63)

1. **Основные типы проекций. Видовая система координат (вывод матрицы преобразования) [1].**

В настоящее время наиболее распространены устройства отображения, которые синтезируют изображение на плоскости – экране дисплея или бумаге.

При использовании любых графических устройств обычно применяются проекции. Проекция задает способ отображения объектов на графическом устройстве. Мы будем рассматривать только проекции на плоскость. Плоскость, на которую выполняется проекция объекта, называется **картинной плоскостью**.

В компьютерной графике наиболее распространены **параллельная** (рис. 10.1а) и **центральная** (рис. 10.1б) проекции.

|  |  |
| --- | --- |
| P  C1  A1  B1  C  B  A | C  O  C1  B1  A  A1  P  B |
| Рис.10.1а | Рис.10.1б |

Для центральной проекции (также называемой перспективной) лучи проецирования исходят из одной точки, расположенной на конечном расстоянии от объектов и плоскости проецирования. Для параллельной проекции лучи проецирования параллельны. Частным случаем параллельной проекции является **аксонометрическая** проекция. Для аксонометрической проекции все лучи проецирования располагаются под прямым углом к плоскости проецирования.

***Видовая система координат***

Для изображения объекта на экране его мировые координаты необходимо преобразовать (пересчитать) в другую систему координат, которая связана с точкой наблюдения. Эта система координат называется видовой системой координат и является левосторонней.

|  |  |
| --- | --- |
|  |  |
| Рис. 10.2 | Рис. 10.3 |

Рассмотрим правую систему мировых координат  (рис. 10.2) и зададим в ней точку наблюдения , где

 (10.1)

Система видовых координат показана на рис. 10.3.

Преобразование мировых координат в видовые можно представить в виде

, (10.2)

где  – координаты некоторой точки в мировой системе координат,  – координаты этой же точки в видовой системе координат, – матрица преобразования.

Получим выражение для матрицы , выполняя переход от мировой системы координат к видовой путем последовательности элементарных преобразований[4].

Сместим начало системы координат  (рис.10.2) в точку  (рис. 10.4), переход →.

|  |  |
| --- | --- |
|  |  |
| Рис. 10.4 | Рис. 10.5 |

Преобразование координат имеет вид

, (10.3)

где

, (10.4)

где .

Повернем систему координат  (рис. 10.4) вокруг оси  на угол , переход  → (рис. 10.5).

Преобразование координат имеет вид

, (10.5)

где .

Поскольку поворот выполняется по часовой стрелке (в отрицательном направлении), то в матрице преобразования  (9.33) следует использовать аргумент . В результате матрица преобразования примет вид

(10.6)

Повернем систему координат  (рис. 10.5) вокруг координатной оси  на угол  , переход →(рис. 10.6)

|  |  |
| --- | --- |
|  |  |
| Рис. 10.6 | Рис. 10.7 |

Преобразование координат имеет вид

, (10.7)

где .

В результате матрица преобразования примет вид

 (10.8)

Изменим направление оси  (рис. 10.6) на противоположное , переход → (рис. 10.7).

Преобразование координат имеет вид

, (10.9)

где

 (10.10)

– матица преобразования для перехода .

Выражение (10.9) с учетом (10.3), (10.5),(10.7)и (10.10) можно представить в виде

, (10.11)

где

. (10.12)

Подставляя в (10.12) выражения для матриц (10.4), (10.6), (10.8) и (10.10), получим



 (10.13)

При получении (10.13) учтено, что элементы матрицы  ,  и  определяются выражениями (10.1).

Таким образом, преобразование (10.2), связывающее координаты мировой и видовой систем координат можно представить в виде

 (10.14)

или

 (10.15)

1. **Перспективные преобразования (вывод матрицы преобразования). Схема пересчета координат при переходе от мировых 3D-координат к экранным [1].**

Рассмотрим, как вычисляются координаты объекта в картинной плоскости, когда лучи проецирования исходят из одной точки, расположенной на конечном расстоянии от объекта (и плоскости проецирования), рис. 10.8.

Пусть  – картинная плоскость в видовой системе координат  (плоскость проецирования – ),  – координаты некоторой точки, полученные пересчетом ее мировых координат в видовую систему координат  без учета перспективы (аксонометрическая проекция) в соответствии с выражениями (10.15). Далее, пусть – проекция точки на картинную плоскость , полученная с учетом перспективы, – - координата точки сходалучей. Необходимо вычислить координаты точки , полагая, что координаты точки являются известными.





























Рис. 10.8

Из подобия треугольников  и  получаем пропорцию

. (10.16)

Откуда находим

. (10.17)

Аналогично из подобия треугольников и  получим

 (10.18)

и

. (10.19)

Вводя обозначения

, (10.20)

можно записать

. (10.21)

Последние выражения можно записать в матричном виде

 (10.22)

или

, (10.23)

где

, ,

. (10.24)

Таким образом, этапы преобразования координат, при переходе от мировых координат к экранным, можно отобразить в виде блок-схемы, которая показана на рис. 10.9.

Сначала мировые координаты преобразуются в видовые с началом в точке  (рис. 10.3). Затем при необходимости выполняется перспективное преобразование, добавляющее эффект перспективы. При построении аксонометрической проекции перспективное преобразование не выполняется и видовые координаты  (рис. 10.9) сразу используются для получения соответствующих оконных координат.

Мировые координаты



Видовые координаты



Нужна перспек-тива?

Перспективные преобразования



Экранные координаты 

Нет

Да

Рис. 10.9

1. **Модели описания поверхностей [1,2]**

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):*

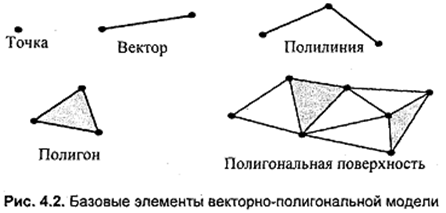


где s и t — параметры, которые изменяются в определенном диапазоне, функции *Fx*, *Fy* и *Fz* будут определять форму поверхности.

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

Векторная полигональная модель

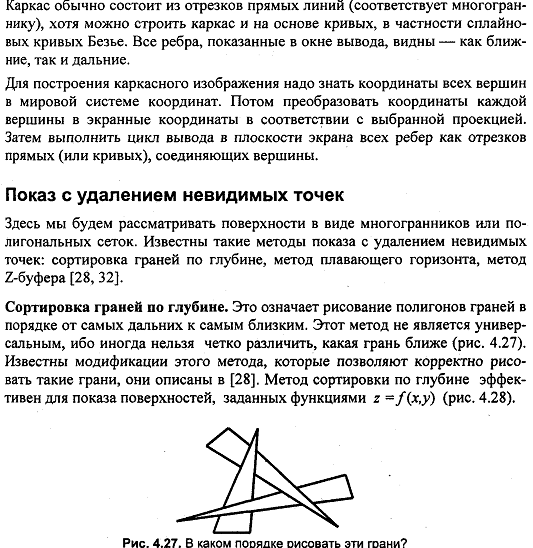
Для описания пространственных объектов здесь используются такие элементы: вершины; отрезки прямых (векторы); полигоны, полигональные поверхности(рис. 4.2).



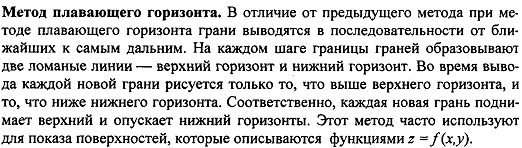
Элемент "вершина" (vertex) – главный элемент описания, все другие являются производными. При использовании трехмерной декартовой систем координаты вершин определяются как(xi, yi, zi).Каждый объект однозначно определяется координатами собственных вершин.

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.



1. **Каркасная визуализация трехмерных изображений. Принцип удаления невидимых граней для выпуклого многогранника. Пример реализации по лабораторной работе. (Листинг функции по лабораторной работе). [1]**



1. **Изображение поверхности z=f(x,y). Метод сортировки граней по глубине (алгоритм художника). Пример реализации метода по лабораторной работе. (Листинг функции по лабораторной работе). [1]**

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.

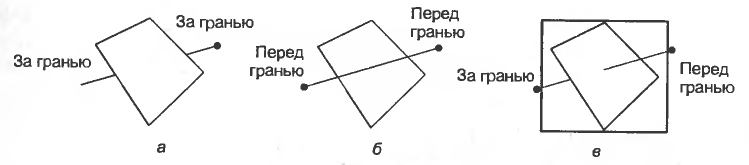
Алгоритм сортировки по глубине используется для удаления невидимых поверхностей. При применении алгоритма невидимых граней к множеству объектов удалено будет лишь около 50% невидимых линий. Нужен алгоритм, который удалял бы все невидимые линии независимо от количества объектов, их выпуклости и наличия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

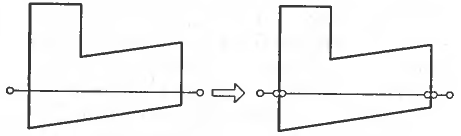
Один из таких алгоритмов действует следующим образом. Для каждого ребра (ребром объекта называется кривая пересечения соседних поверхностей, ограничивающих внутренний объем объекта) каждого объекта производится проверка, не закрыто ли оно гранями (гранями называются поверхности, ограничивающие объем 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.





1. **Закрашивание поверхностей. Модели отражения света [1,2]**

Рассмотрим, как можно определить цвет пикселов изображения поверхности согласно интенсивности отраженного света при учете взаимного расположения поверхности, источника света и наблюдателя.

***Зеркальное отражение света***

Угол между нормалью и падающим лучом равен углу между нормалью и отраженным лучом. Падающий луч, отраженный, и нормаль располагаются в одной плоскости (рис. 13.1).







Нормаль

Наблюдатель

Поверхность

Источник света

Отраженный луч

Рис. 13.1

Поверхность считается **идеально зеркальной**, если на ней отсутствуют какие-либо неровности, шероховатости. Собственный цвет у такой поверхности не наблюдается. Световая энергия падающего луча отражается только по линии отраженного луча. Какое-либо рассеяние в стороны от этой линии отсутствует. В природе, вероятно, нет идеально гладких поверхностей, поэтому полагают, что если глубина шероховатостей существенно меньше длины волны излучения, то рассеивания не наблюдается. Для видимого спектра можно принять, что глубина шероховатостей поверхности зеркала должна быть существенно меньше 0.5 мкм [2].

Если поверхность зеркала отполирована неидеально, то наблюдается зависимость интенсивности отраженного света от длины волны – чем больше длина волны, тем лучше отражение. Например, красные лучи отражаются сильнее, чем синие.

Падающий луч, попадая на слегка шероховатую поверхность реального зеркала, порождает не один отраженный луч, а несколько лучей, рассеиваемых по различным направлениям. Зона рассеивания зависит от качества полировки и может быть описана некоторым законом распределения. Как правило, форма зоны рассеивания симметрична относительно линии идеального зеркально отраженного луча. К числу простейших, но достаточно часто используемых, относится эмпирическая модель распределения Фонга, согласно которой интенсивность зеркально отраженного излучения определяется выражением

, (13.1)

где – интенсивность излучения источника,– коэффициент пропорциональности,  – угол отклонения от линии идеально отраженного луча (рис. 13.1),показатель  находится в диапазоне от 1 до 200 и зависит от качества полировки.

***Диффузное отражение света***

Этот вид отражения присущ **матовым** поверхностям. Матовой можно считать такую поверхность, размер шероховатостей которой уже настолько велик, что падающий луч рассеивается равномерно во все стороны. Такой тип отражения характерен, например, для гипса, песка, бумаги.

Для матовой поверхности законы отражения установлены Ламбертом. Их суть иллюстрируется рис. 13.2 и 13.3.

|  |  |
| --- | --- |
|  |  |
| Рис. 13.2 | Рис. 13.3 |

Если световой поток интенсивностью (мощностью)  падает нормально к матовой поверхности (рис. 13.2), то интенсивность вторичного излучения под углом  к нормали, проведенной к элементарной площадке , пропорционально [2].

, (13.2)

где  – коэффициент, который учитывает свойства материала поверхности. Значение  находится в диапазоне от 0 до 1.

Физический смысл формулы (13.2) заключается в том, что в направлении, определяемом углом , площадь  проектируется как , при этом пропорционально  уменьшается площадь излучающей площадки и интенсивность излучения.

Если поток излучения падает под углом  по отношению к нормали, проведенной к поверхности в месте падения (рис.13.3), то

,где .

Следовательно

. (13.3)

Здесь  – интенсивность светового потока, которую фиксировал бы приемник, находящийся в «зените», если бы площадка облучалась бы то же с зенита.

Физический смысл формулы (13.3) состоит в том, что с увеличением  уменьшается перехватываемый поверхностью падающий световой поток, отчего уменьшается ее освещенность и, как следствие, яркость.

Матовая поверхность имеет свой цвет. Наблюдаемый цвет матовой поверхности определяется комбинацией собственного цвета поверхности и цвета излучения источника света.

При создании реалистичных изображений следует учитывать то, что в природе, вероятно, не существует идеально зеркальных или полностью матовых поверхностей. При изображении объектов средствами компьютерной графики обычно моделируют сочетание зеркальности и диффузного рассеивания в пропорции, характерной для конкретного материала. В этом случае модель отражения записывают в виде суммы диффузной и зеркальной компонент:

, (13.4)

где константы  и , определяют отражательные свойства материала. Согласно этой формуле интенсивность отраженного света равна нулю для некоторых углов  и . Однако в реальных сценах обычно нет полностью затемненных объектов, следует учитывать фоновую подсветку, освещение рассеянным светом, отраженным от других объектов. В таком случае интенсивность может быть эмпирически выражена следующей формулой:

, (13.5)

где– интенсивность рассеянного света,**константа.

Можно еще усовершенствовать модель отражения, если учесть то, что энергия от точечного источника света уменьшается пропорционально квадрату расстояния. Использование такого правила вызывает сложности, поэтому на практике часто реализуют модель, выражаемую эмпирической формулой[2]:

, (13.6)

где– расстояние от центра проекции до поверхности,константа.

Распределение световой энергии по возможным направлениям световых лучей можно отобразить с помощью векторных диаграмм, в которых длина векторов соответствует интенсивности (рис. 13.4). На приведенном рисунке показаны векторные диаграммы для различных видов отражений: *а*) – идеальное зеркальное, *б*) – неидеальное зеркальное, *в*) – диффузионное, *г*) – сумма диффузионного и зеркального.

|  |  |
| --- | --- |
| *а* | *б* |
| *в* | *г* |

Рис. 13.4

Рассмотрим, как определить цвет закрашивания точек объектов в соответствии с выбранной моделью. Наиболее просто выполняется расчет в градациях серого цвета (например, для белого источника света и серых объектов). В данном случае интенсивность отраженного света соответствует яркости. Сложнее обстоит дело с цветными источниками света, освещающими цветные поверхности. Например, для модели RGB составляются три формулы расчета интенсивности отраженного света для различных цветовых компонент. Коэффициенты**и ** различны для разных компонент – они выражают собственный цвет поверхности. Поскольку цвет отраженного зеркального луча равен цвету источника, то коэффициент , будет одинаковым для всех компонент цветовой модели. Цвет источника света выражается значениями интенсивности для соответствующих цветовых компонент.

1. **Закрашивание поверхностей. Вычисление нормалей и углов отражения. (Листинг функции для пирамиды по лабораторной работе)**

В рассматриваемых нами моделях отражения света присутствуют такие параметры как угол падения и отражения , угол  – угол отклонения от линии идеально отраженного луча (рис. 13.1) и угол  – угол под которым наблюдается результат диффузионного отражения света (рис. 13.2, рис.13.3).

Рассмотрим алгоритмы вычисления этих углов в выбранной системе координат.

**Диффузионное отражение.** Определим косинус угла  между вектором нормали  к поверхности и некоторым направлением, определяемым вектором . Вектор  может указывать как на положение источника света, так и на положение точки наблюдения. Таким образом, угол  – это угол, соответствующий углам или  на рис. 13.1–13.3.

Сначала рассмотрим случай, когда источник света или наблюдатель находятся на бесконечности по отношению к некоторому элементу поверхности (рис. 13.5).

Пусть заданы:

 – вектор нормали к элементу поверхности и

 – вектор, определяющий некоторое направление в пространстве.

Тогда

. (13.7)











Рис. 13.5

Перейдем к рассмотрению случая, когда источник света или наблюдатель находятся на конечном расстоянии от поверхности (рис. 13.6)

















Рис. 13.6

Пусть заданы в системе координат  (рис. 13.6):

 – вектор нормали к элементу поверхности,

 – радиус-вектор, определяющий положение источника светаили точки наблюдения,

 – радиус-вектор, определяющий положение элемента поверхности.

Тогда

 (13.8)

Координаты вектора  определяются из рис. 13.6

 (13.9)

Таким образом

 (13.10)

С учетом (13.9) и (13.10) выражение (13.8) можно представить в виде

(13.11)

**Зеркальное отражение**

Пусть заданы (рис. 13.7)

 – вектор нормали к элементу поверхности,

 – вектор, определяющий направление на источник света,

 – вектор, направление отраженного луча,

 – вектор, определяющий направление на камеру.

Будем полагать, что

. (13.12)





















Рис. 13.7

Требуется определить косинус угла .

На векторах ,  и  построим соответствующие им единичные вектора , ,  и выполним построения, показанные на рис. 13.7.

, ,  (13.13)

Из рис. 13.7 получаем

, (13.14)

, (13.15)

где символ «» – знак скалярного произведения.

Подставляя (13.15) в (13.14), получим

. (13.16)

Отсюда

. (13.17)

Подставляя в (13.17) выражения для единичных векторов из (13.13), получим

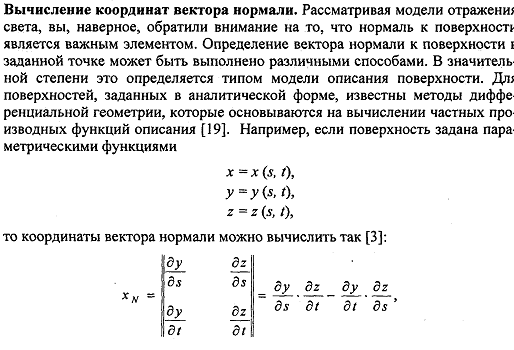
,

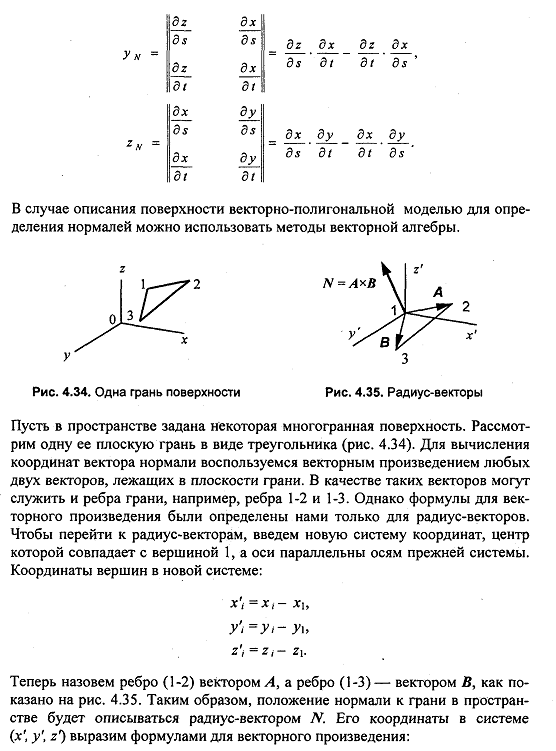
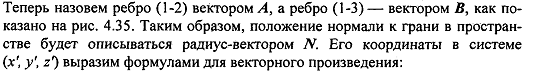
откуда с учётом (13.12)

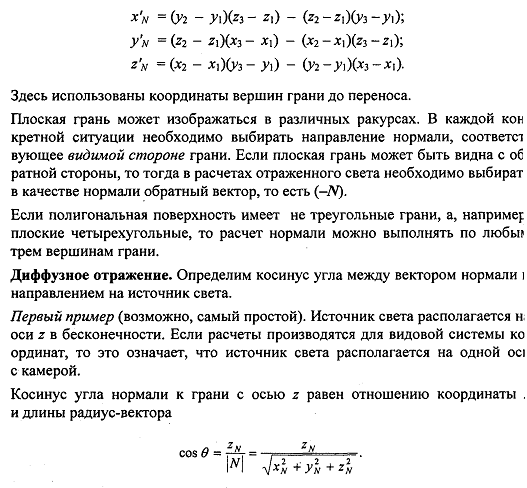
 (13.18)

И окончательно

. (13.19)

****

****  




**Листинг функций**

void CPyramid::ColorDraw(CDC& dc, CMatrix& PView, CRect& RW, COLORREF Color)

{

BYTE red = GetRValue(Color); //получаем битовые значения цветов для переданного в функцию цвета

BYTE green = GetGValue(Color);

BYTE blue = GetBValue(Color);

CMatrix ViewCart = SphereToCart(PView); // Преобразует сферические координаты PView точки в декартовы (декартовы координаты точки обзора)

CMatrix MV = CreateViewCoord(PView(0), PView(1), PView(2)); // Матрица пересчёта

CMatrix ViewVert = MV\*Vertices; // Переводим значения

CRectD RectView;

GetRect(ViewVert, RectView); // Устанавливает область определения

CMatrix MW = SpaceToWindow(RectView, RW); // Возвращает матрицу пересчета координат из мировых в оконные

CPoint MasVert[6]; //Массив точек для отрисовки

CMatrix V(3);

V(2) = 1;

for (int i = 0; i<6; i++)

{

V(0) = ViewVert(0, i); // x

V(1) = ViewVert(1, i); // y

//V=V-V0; // x-xL, y-yL, 1

V = MW\*V;

MasVert[i].x = (int)V(0);

MasVert[i].y = (int)V(1);

}

CMatrix VE = Vertices.GetCol(4, 0, 2); // Возвращает подстолбец из столбца матрицы с номером k

// n - номер первого элемента в столбце

// m - номер последнего элемента в столбце

CMatrix R1(3), R2(3), VN(3);

double sm;

for (int i = 0; i<3; i++) // Вершины

{

int k;

if (i == 2) k = 0; // Сбрасываем для того, чтобы перескакивать

else k = i + 1;

R1 = Vertices.GetCol(i, 0, 2);

R2 = Vertices.GetCol(k, 0, 2);

CMatrix V1 = R2 - R1; // Вектор - ребро в основании

CMatrix V2 = VE - R1; // Вектор - ребро к вершине E

VN = VectorMult(V2, V1); // Вычисляет векторное произведение векторов V1 и V2 (Вектор ВНЕШНЕЙ(!)нормали к грани)

sm = CosV1V2(VN, ViewCart); // Вычисляет КОСИНУС угла между векторами VN и ViewCart - точка просмотра

if (sm >= 0) //Грань винима - рисуем боковую грань

{

CPen Pen(PS\_SOLID, 2, RGB(sm\*sm\*red, sm\*sm\*green, sm\*sm\*blue));

CPen\* pOldPen = dc.SelectObject(&Pen);

CBrush Brus(RGB(sm\*sm\*red, sm\*sm\*green, sm\*sm\*blue));

CBrush\* pOldBrush = dc.SelectObject(&Brus);

CPoint MasVertR[4] = { MasVert[k], MasVert[k + 3], MasVert[i + 3], MasVert[i] }; // Делаем набор точек грани

// На основе набора координат

dc.Polygon(MasVertR, 4); // Рисуем полигон-верхнее основание

dc.SelectObject(pOldBrush);

dc.SelectObject(pOldPen);

}

}

VN = VectorMult(R1, R2); // Вычисляет векторное произведение векторов R1 и R2

if (ViewCart(2)<0) // Основания

{

CPen Pen(PS\_SOLID, 2, RGB(sm\*sm\*red, sm\*sm\*green, sm\*sm\*blue));

CPen\* pOldPen = dc.SelectObject(&Pen);

CBrush Brus(RGB(sm\*sm\*red, sm\*sm\*green, sm\*sm\*blue));

CBrush\* pOldBrush = dc.SelectObject(&Brus);

dc.Polygon(MasVert, 3); // Основание

dc.SelectObject(pOldBrush);

dc.SelectObject(pOldPen);

}

else

{

CPen Pen(PS\_SOLID, 2, RGB(sm\*sm\*red, sm\*sm\*green, sm\*sm\*blue));

CPen\* pOldPen = dc.SelectObject(&Pen);

CBrush Brus(RGB(sm\*sm\*red, sm\*sm\*green, sm\*sm\*blue));

CBrush\* pOldBrush = dc.SelectObject(&Brus);

dc.Polygon(MasVert + 3, 3); // Основание

dc.SelectObject(pOldBrush);

dc.SelectObject(pOldPen);

}

}

void CPyramid::GetRect(CMatrix& Vert,CRectD& RectView)

{// Вычисляет координаты прямоугольника,охватывающего проекцию пирамиды

// на плоскость XY в ВИДОВОЙ системе координат

// Vert - координаты вершин (в столбцах)

// RectView - проекция - охватывающий прямоугольник

CMatrix V=Vert.GetRow(0);

double xMin=V.MinElement();

double xMax=V.MaxElement();

V=Vert.GetRow(1);

double yMin=V.MinElement();

double yMax=V.MaxElement();

RectView.SetRectD(xMin,yMax,xMax,yMin);

}

1. **Метод закрашивания Гуро [1,2].**

Этот метод предназначен для создания иллюзии гладкой криволинейной поверхности, описанной в виде многогранников или полигональной сетки с плоскими гранями. Если каждая плоская грань имеет один постоянный цвет, определенный с учетом отражения, то различные цвета соседних граней очень заметны, и поверхность выглядит именно как многогранник. Казалось бы, этот дефект можно замаскировать за счет увеличения количества граней при аппроксимации поверхности. Но зрение человека имеет способность подчеркивать перепады яркости на границах смежных граней – такой эффект называется **эффектом полос Маха***.* Поэтому для создания иллюзии гладкости нужно намного увеличить количество граней, что приводит к существенному замедлению визуализации – чем больше граней, тем меньше скорость рисования объектов.

Метод Гуро основывается на идее закрашивания каждой плоской грани не одним цветом, а плавно изменяющимися оттенками, вычисляемыми путем интерполяции цветов примыкающих граней. Закрашивание граней по методу Гуро осуществляется в четыре этапа:

– вычисляются нормали к каждой грани;

– определяются нормали в вершинах, нормаль в вершине определяется усреднением нормалей примыкающих граней (рис. 13.8);

– на основе нормалей в вершинах вычисляются значения интенсивностей в вершинах согласно выбранной модели отражения света;

– закрашиваются полигоны граней цветом, соответствующим линейной интерполяции значений интенсивности в вершинах.

Вектор нормали в вершине а (рис. 13.8) равен

 (13.20)





b



c

a



Рис. 13.8

Определение интерполированных значений интенсивности отраженного света в каждой точке грани (и, следовательно, цвет каждого пиксела) удобно выполнять во время цикла заполнения полигона. Рассмотрим заполнение контура грани горизонталями в экранных координатах (рис. 13.9).

Горизонталь заполнения

























Рис. 13.9

Интерполированная интенсивность  в точке  определяется исходя из пропорции

. (13.21)

Отсюда находим

. (13.22)

Значения интенсивностей  и на концах горизонтального отрезка представляют собой интерполяцию интенсивности в вершинах и могут быть получены из пропорций:





Откуда получаем

 (13.23)

. (13.24)

1. Графическая библиотека OpenGL. Особенности, основные возможности, архитектура, синтаксис команд

**OpenGL** является одним из самых популярных прикладных программных интерфейсов (API – **A**pplication **P**rogramming **I**nterface) для разработки приложений в области двумерной и трехмерной графики [9 – 11].Стандарт OpenGL (**O**pen **G**raphics **L**ibrary – открытая графическая библиотека) был разработан и утвержден в 1992 году ведущими фирмами в области разработки программного обеспечения как эффективный аппаратно-независимый интерфейс, пригодный для реализации на различных платформах. Основой стандарта стала библиотека IRISGL, разработанная фирмой Silicon Graphics Inc. Библиотека насчитывает около 120 различных команд, которые программист использует для задания объектов и операций, необходимых для написания интерактивных графических приложений. На сегодняшний день графическая система OpenGL поддерживается большинством производителей аппаратных и программных платформ. Эта система доступна тем, кто работает в среде Windows, пользователям компьютеров Apple. Характерными особенностями OpenGL, которые обеспечили распространение и развитие этого графического стандарта, являются:

* **Стабильность**. Дополнения и изменения в стандарте реализуются таким образом, чтобы сохранить совместимость с разработанным ранее программным обеспечением.
* **Надежность и переносимость**. Приложения, использующие OpenGL, гарантируют одинаковый визуальный результат вне зависимости от типа используемой операционной системы и организации отображения информации. Кроме того, эти приложения могут выполняться как на персональных компьютерах, так и на рабочих станциях и суперкомпьютерах.
* **Легкость применения**. Стандарт OpenGL имеет продуманную структуру и интуитивно понятный интерфейс, что позволяет с меньшими затратами создавать эффективные приложения, содержащие меньше строк кода, чем с использованием других графических библиотек. Необходимые функции для обеспечения совместимости с различным оборудованием реализованы на уровне библиотеки и значительно упрощают разработку приложений.

***Основные возможности***

Описать возможности OpenGL можно через функции его библиотеки. Все функции можно разделить на пять категорий:

1. **Функции описания примитивов** определяют объекты нижнего уровня иерархии (примитивы), которые способна отображать графическая подсистема. В OpenGL в качестве примитивов выступают точки, линии, многоугольники и т.д.
2. **Функции описания источников света** служат для описания положения и параметров источников света, расположенных в трехмерной сцене.
3. **Функции задания атрибутов**. С помощью задания атрибутов программист определяет, как будут выглядеть на экране отображаемые объекты. Другими словами, если с помощью примитивов определяется, **что** появится на экране, то атрибуты определяют **способ** вывода на экран. В качестве атрибутов OpenGL позволяет задавать цвет, характеристики материала, текстуры, параметры освещения.
4. **Функции визуализации** позволяет задать положение наблюдателя в виртуальном пространстве, параметры объектива камеры. Зная эти параметры, система сможет не только правильно построить изображение, но и отсечь объекты, оказавшиеся вне поля зрения.
5. **Функции геометрических преобразований** позволяют программисту выполнять различные преобразования объектов – поворот, перенос, масштабирование. При этом OpenGL может выполнять дополнительные операции, такие как использование сплайнов для построения линий и поверхностей, удаление невидимых фрагментов изображениями на уровне пикселей и т.д.

***Архитектура OpenGL***

Функции OpenGL реализованы в модели клиент–сервер. Приложение выступает в роли клиента – оно вырабатывает команды, а сервер OpenGL интерпретирует и выполняет их. Сам сервер может находиться как на том же компьютере, на котором находится клиент (например, в виде динамически загружаемой библиотеки – DLL), так и на другом (при этом может быть использован специальный протокол передачи данных между машинами).

GL обрабатывает и рисует в буфере кадра графические примитивы с учетом некоторого числа выбранных режимов. Каждый примитив – это точка, отрезок, многоугольник и т. д. Каждый режим может быть изменен независимо от других. Определение примитивов, выбор режимов и другие операции описываются с помощью команд в форме вызовов функций прикладной библиотеки.

Примитивы определяются набором из одной или более вершин (**vertex**). Вершина определяет точку, конец отрезка или угол многоугольника. С каждой вершиной ассоциируются некоторые данные (координаты, цвет, нормаль, текстурные координаты и т. д.), называемые атрибутами. В подавляющем большинстве случаев каждая вершина обрабатывается независимо от других.

С точки зрения архитектуры графическая система OpenGL является конвейером (рис. 15.2), состоящим из нескольких последовательных этапов обработки графических данных.

Команды OpenGL всегда обрабатываются в том порядке, в котором они поступают, хотя могут происходить задержки перед тем, как проявится эффект от их выполнения. В большинстве случаев OpenGL предоставляет непосредственный интерфейс, т. е. определение объекта вызывает его визуализацию в буфере кадра.

С точки зрения разработчиков, OpenGL – это ***набор команд, которые управляют использованием графической аппаратуры***. Если аппаратура состоит только из адресуемого буфера кадра, тогда OpenGL должен быть реализован полностью с использованием ресурсов центрального процессора. Обычно графическая аппаратура предоставляет различные уровни ускорения: от аппаратной реализации вывода линий и многоугольников до изощренных графических процессоров с поддержкой различных операций над геометрическими данными.

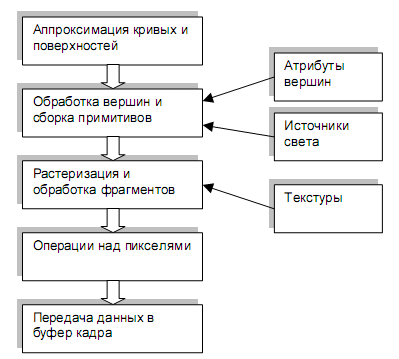


Рис. 15.2

OpenGL является прослойкой между аппаратурой и пользовательским уровнем, что позволяет предоставлять единый интерфейс на разных платформах, используя возможности аппаратной поддержки.

Кроме того, OpenGL можно рассматривать как конечный автомат, состояние которого определяется множеством значений специальных переменных и значениями текущей нормали, цвета, координат текстуры и других атрибутов и признаков. Вся эта информация будет использована при поступлении в графическую систему координат вершины для построения фигуры, в которую она входит. Смена состояний происходит с помощью команд, которые оформляются как вызовы функций.

***Синтаксис команд***

Определения команд GL находятся в файле gl.h, для включения которого нужно написать (для Windows 7)

**#include<gl.h>**

Для работы с библиотекой GLU нужно аналогично включить файл glu.h. Версии этих библиотек, как правило, включаются в дистрибутивы систем программирования, например Microsoft Visual Studio, Borland Delphi.

В отличие от стандартных библиотек, пакет GLUT нужно инсталлировать и подключать отдельно.

Все команды (процедуры и функции) библиотеки GL начинаются с префикса **gl**, все константы – с префикса **GL\_**. Соответствующие команды и константы библиотек GLUи GLUT аналогично имеют префиксы **glu** (GLU\_) и **glut** (GLUT\_).

Кроме того, в имена команд входят суффиксы, несущие информацию о числе и типе передаваемых параметров. В OpenGL полное имя команды имеет вид:

**type LibNameCommandName[1 2 3 4][b s i f d ub us ui][v]**

**(type1 arg1,…,typeNargN)**,

где

**type**– тип возвращаемого функцией значения,

**LibNameCommandName**– имя команды, которое состоит из двух частей:**LibName**и**CommandName**,

**LibName** – имя библиотеки, в которой описана эта функция

Для базовых функций OpenGL, функций из библиотек GL, GLU, GLUT, GLAUX это **gl**, **glu**, **glut**, **aux** соответственно.

**CommandName –**имя команды (процедуры или функции),

**[1 2 3 4] –** число аргументов команды,

**[b s i f d ub us ui] –** тип аргумента, символ.

Соответствие типов OpenGL и языка C(C++) приведено в табл. 15.1.

Таблица 15.1

**Типы OpenGL и языка C(C++)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Символ** | **Описание** | **Тип OpenGL** | **Тип C( C++)** |
| b | 8 – битовое целое | GLbyte | signed char |
| s | 16 – битовое целое | GLshort | short |
| i | 32 – битовое целое | GLint  Gsizei | long |
| f | 32 – битовое число с плавающей точкой | GLfloat  GLclampf | float |
| d | 64 – битовое число с плавающей точкой | GLdouble  GLclampf | double |
| ub | 8 – битовое беззнаковоецелое | GLubyte  GLboolean | unsigned char |
| us | 16 – битовое беззнаковое целое | GLushort | unsigned short |
| ui | 32 – битовое беззнаковое целое | GLuint  GLenum  GLbitfield | unsigned long |
|  |  | GLvoid | void |

**h [v] –** наличие этого символа показывает, что в качестве параметров функции используется указатель на массив значений. Символы в квадратных скобках в некоторых названиях не используются. Например, команда **glVertex2i()**, описанная в библиотеке GL, использует в качестве параметров два целых числа, а команда **glColor3fv()** использует в качестве параметра указатель на массив из трех вещественных чисел. Использования нескольких вариантов каждой команды можно частично избежать, применяя перегрузку функций языка C++. Но интерфейс OpenGL не рассчитан на конкретный язык программирования, и, следовательно, должен быть максимально универсален.

1. Графическая библиотека OpenGL. Структура GLUT-приложения. Пример
2. Графическая библиотека OpenGL. Рисование графических примитивов. Дисплейные списки. Удаление нелицевых граней. Вывод текста. Примеры
3. Графическая библиотека OpenGL. Системы координат. Модельно-видовые преобразования. Примеры
4. Графическая библиотека OpenGL. Проекции. Область вывода. Создание анимации. Примеры
5. Графическая библиотека OpenGL. Материалы и освещение. Примеры
6. Графическая библиотека OpenGL. Структура приложения с использованием библиотеки MFC