1. **Цели и задачи компьютерной графики**

Понятие «компьютерная графика» объединяет довольно широкий круг опе­раций по обработке графической информации с помощью компьютера. Причем наблюдается явная тенденция «компьютеризации» изображений, циркулирующих в обществе.

В компьютерной графике можно выделить несколько основных на­правлений:

* **Визуализация научных данных.**

Большинство современных математических программных пакетов (на­пример. Maple, MatLab. MaihCAD) имеют средства для отображения гра­фиков, поверхностей и трехмерных тел, построенных на основе каких-либо расчетов.

* 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.

Возможно, самый известный пример распознавания образов — сканиро­вание и перевод "фотографии'' текста в набор отдельных символов, фор­мирующих слова.

* Виртуальная реальность.

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.

1. **Растровая и векторная визуализация изображений. Основные характеристики растровых изображений**

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.

Характеристики растровых изображений:

**Разрешающая способность**. Она характеризует расстояние между соседни­ми пикселами (рис. 1). 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).

**Размер** растра обычно измеряется количеством пикселов по горизонтали и вертикали.

*Форма пикселов* растра определяется особенностями устройства графиче­ского вывода. 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).

1. **Аддитивная цветовая модель RGB. Кодирование цвета. Палитра**

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

**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. 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).

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)**

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

Форматы:

***Векторный формат*** наиболее удобен для хранения изображений, которые можно разложить на простые геометрические фигуры (например, чертежи или текст).

***Растровый формат*** используется для хранения растровых данных.

*Битовые изображения*, как правило, выводятся на экран быстрее, так как их внутренняя структура аналогична (до некоторой степени) структуре видеопамяти.

***Метафайловый формат*** позволяет хранить в одном файле и векторные, и растровые данные.

***Битовое изображение DDB есть набор бит в оперативной памяти, который может быть отображен на устройстве вывода.*** 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**. 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**.

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

**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

, , , 

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

**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.

1. **Стиль линии (перо). Алгоритмы вывода линий**

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.

Алгоритмы вывода:

* ***Вывод пиксела (х, у)*** Можно представить себе такой алгоритм, как цикл, в котором определяются координаты  каждого пиксела.
* ***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.

1. **Стиль заполнения, кисть, текстура**

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.

***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. 

***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

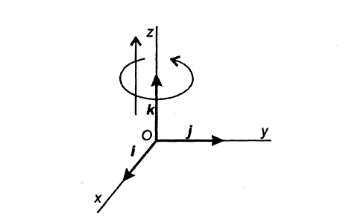
1. ***Геометрические основы компьютерной графики*. Системы координат (декартова, полярная, сферическая) и векторы. Скалярное и векторное произведение векторов**

Полярная система координат

|  |  |
| --- | --- |
| Сферическая система координат |  |

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

Начальная точка векторов обозначается буквой О.



Пусть заданы два вектора и 

 и 

Тогда



При этом справедливы равенства



Скалярное произведение векторов

,

где  - угол между векторами и 

В координатной форме





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: 

**Векторное произведение векторов**

Векторное произведение векторов  и  определяется выражением

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

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





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**

Пусть в декартовой системе координат на плоскости  заданы две точки

 и 

Тогда уравнение прямой, проходящей через эти точки имеет вид:

 (7.20)

Пологая в (7.20)

,

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

, (7.21)

Пусть



Тогда (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 .

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

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 ,

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.

**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. (Листинг функции для реализации кубического сплайна по лабораторной работе).**

Сплайном называется кусочно-полиномиальная функция, определенная наотрезке [a, b] и имеющая на этом отрезке некоторое количество непрерывных производных. Преимущества интерполяции сплайнами по сравнению с обычными методами интерполяции – в сходимости и устойчивости вычислительного процесса.

Рассмотрим один из наиболее распространенных в практике случаев – интерполирование функции кубическим сплайном.  
Пусть на отрезке [a, b] задана непрерывная функцияhttp://www.simumath.net/library/materials/Interpol_splines/images/Eqn001.png. Введем разбиение отрезка:

http://www.simumath.net/library/materials/Interpol_splines/images/Eqn003.png          (6)

и обозначим http://www.simumath.net/library/materials/Interpol_splines/images/Eqn004.png , http://www.simumath.net/library/materials/Interpol_splines/images/Eqn030.png.

Сплайном, соответствующим данной функцииhttp://www.simumath.net/library/materials/Interpol_splines/images/Eqn006.pngи узлам интерполяции (6) называется функция http://www.simumath.net/library/materials/Interpol_splines/images/Eqn010.png, удовлетворяющая следующим условиям:

1) на каждом отрезке  http://www.simumath.net/library/materials/Interpol_splines/images/Eqn007.png,http://www.simumath.net/library/materials/Interpol_splines/images/Eqn008.png  функция http://www.simumath.net/library/materials/Interpol_splines/images/Eqn010.png является кубическим многочленом;

2) функция http://www.simumath.net/library/materials/Interpol_splines/images/Eqn010.png, а также ее первая и вторая производные непрерывны на отрезке [a, b] ;

3) http://www.simumath.net/library/materials/Interpol_splines/images/Eqn011.png

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 является 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)**

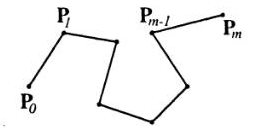
Во многих задачах требование того, чтобы конструируемая кривая однозначно проектировалась соответственно на прямую, является слишком жестким. Расширяя допустимые классы кривых, естественно обратиться и к более общему способу описания их частичных фрагментов. В качестве нового способа задания кривых удобно использовать парамет­рический способ.

Формулировка задачи: по заданному множеству вершин



с 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.



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 внутреннейлибо 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:

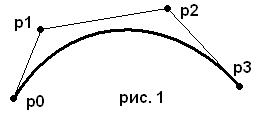
 (1)

где  - некоторые функциональные коэффициенты, подлежащие определению.

Кривая Безье

Кривая Безье задается несколькими точками в N-мерном пространстве (классический вариант, описываемый в литературе – 4 точки, но мы, для удобства, помимо этого случая будем использовать вырожденные случаи из 3-х точек, а так же пользоваться введением дополнительных вершин), но нам так много не нужно, поэтому ограничимся тремя измерениями (хотя все приведенные алгоритмы будут справедливы и для векторов другой мерности).

Для начала рассмотрим самый простой случай (его иллюстрация этого случая приведена на рис. 1) – когда у нас 4 вершины.



Здесь p0 и p3 – начальная и конечная точки соответственно, а p1 и p2 – промежуточные точки. Разности же вершин p1-p0 и p2-p3 определяют крутизну кривой в этих точках. Сама же кривая задается следующим уравнением:

C(s) = p0 \* ( 1 – s )3 + 3 \* p1 \* s \* ( 1 – s )2 + 3 \* p2 \* s2 \* ( 1 – s ) + p3 \* s3

Несложно показать, что

C( 0 ) = p0

C( 1 ) = p3

C’( 0 ) = 3 \* ( p1 - p0 )

C’( 1 ) = 3 \* ( p3 – p2 )

Собственно как я и говорил – кривая проходит через конечные точки + имеется информация о крутизне кривой.

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

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

Экранные – координаты на экране, всегда двумерные.













0









P



0























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 |



Или



, 



или



где

, , 

Перепишем пару рассматриваемых равенств в виде





,

, , 

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

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

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

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

Физическая система координат

На рис. показана физическая система координат для экрана видеомонитора.

(0,0)

X

X

(x,y)

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

Логическая система координат

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

Y

Y

(x,y)

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

int SetMapMode(HDC hdc, int nMapMode);

Параметр nMapMode может принимать одно из следующих значений.

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

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

int GetMapMode(HDC hdc);

Преобразование координат

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





Org – Origin – начало координат

В этих формулах используются следующие обозначения:

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

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

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

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

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

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

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

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

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

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

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Режим отображения** | **Направление оси X** | **Направление**  **оси Y** | **Размер одной логической единицы** |
| MM\_TEXT | Вправо | Вниз | 1 пиксел |
| 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.

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.

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

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

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. 1а, рис 1б).

|  |  |
| --- | --- |
|  |  |
| Рис. 1а | Рис. 1б |

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

 ,

где 

Выражение (1) представляет собой аффинное преобразование координат при переходе от системы координат к системе координат .

Обратный переход от СК к СК определяется как

,

Аффинное преобразование (1) удобно представить в матричном виде:



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



или

,

где

, , 

Обратное преобразование

,

где



Параллельный сдвиг системы координат

,



Или



|  |  |
| --- | --- |
|  |  |
| Рис. 2а | Рис. 2б |

Здесь



Обратное преобразование

,

,

где



Растяжение – сжатие системы координат





Или

,

где



Обратное преобразование

,



Или

,

где

,

Поворот системы координат

|  |  |
| --- | --- |
|  |  |
| Рис. 3а | Рис. 3б |

Координаты точки  в системе координат :



В системе координат :



С учетом (23), получаем:



Или



,

где



Обратное преобразование

Обратное преобразование соответствует повороту системы координат

 на угол 



Или



,

где



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

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

Пусть в системе координат  некоторая точка перемещается из положения  в положение (рис. 4а, 4б) или обратно из положения  в положение .

|  |  |
| --- | --- |
|  |  |
| Рис. 4а | Рис. 4б |

Рассмотрим частные случаи аффинных преобразований объектов.

Сдвиг объекта

|  |  |
| --- | --- |
|  |  |
| Рис. 5 | Рис. 6 |

Из рис. 5 получаем

,



Или



Здесь



Обратное преобразование

Обратное преобразование позволяет вычислить старые координаты объекта  по новым 

,

,

где



Растяжение – сжатие объекта





Или

,

где



Обратное преобразование

,



Или

,

где

,

Поворот объекта вокруг центра координат

Координаты точки  в системе координат (рис. 6):



Координаты точки  в системе координат :



С учетом (46), получаем:



Или



,

где



Обратное преобразование

Обратное преобразование можно получить, если представить себе поворот объекта с координатами  на угол .



Или



,

где



Связь преобразований объектов с преобразованиями координат

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

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

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

**,**

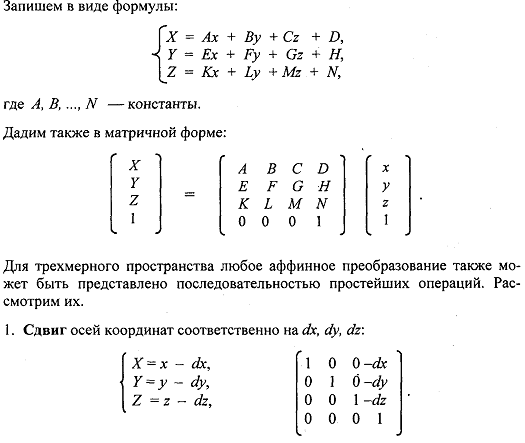
выражений (18) и (42) для растяжения – сжатия

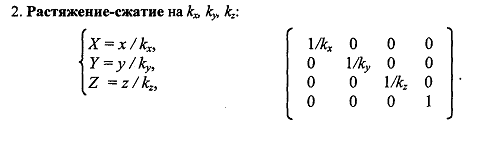


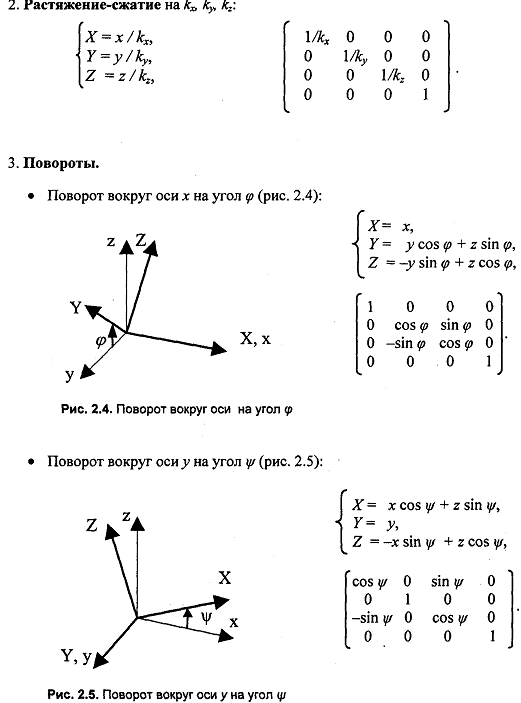
выражений (27) и (51) для поворота

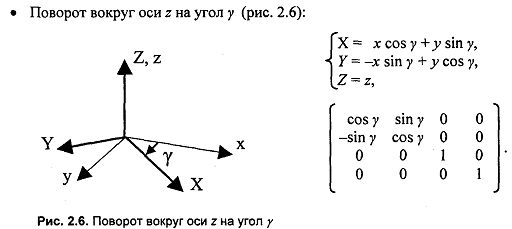


## 19. Аффинные преобразования в пространстве. Преобразование системы координат: смещение, растяжение-сжатие, повороты вокруг осей координат

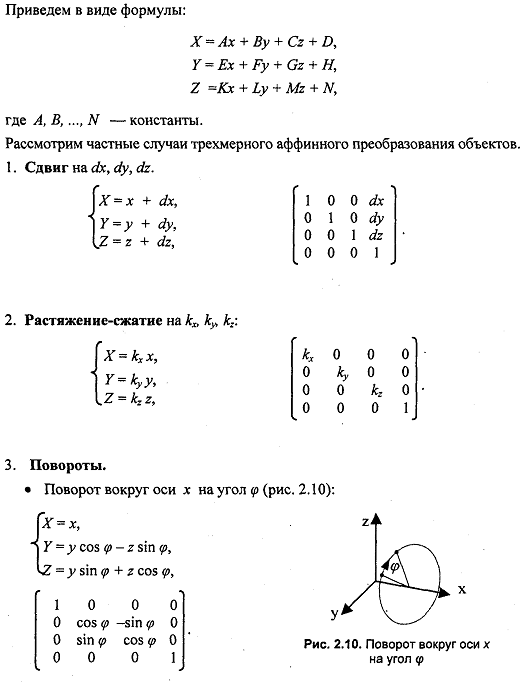


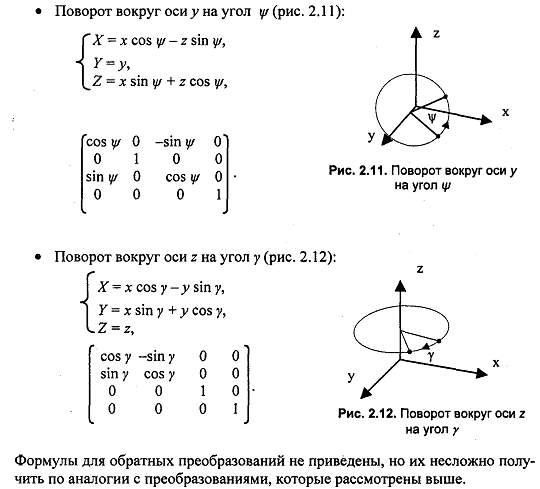






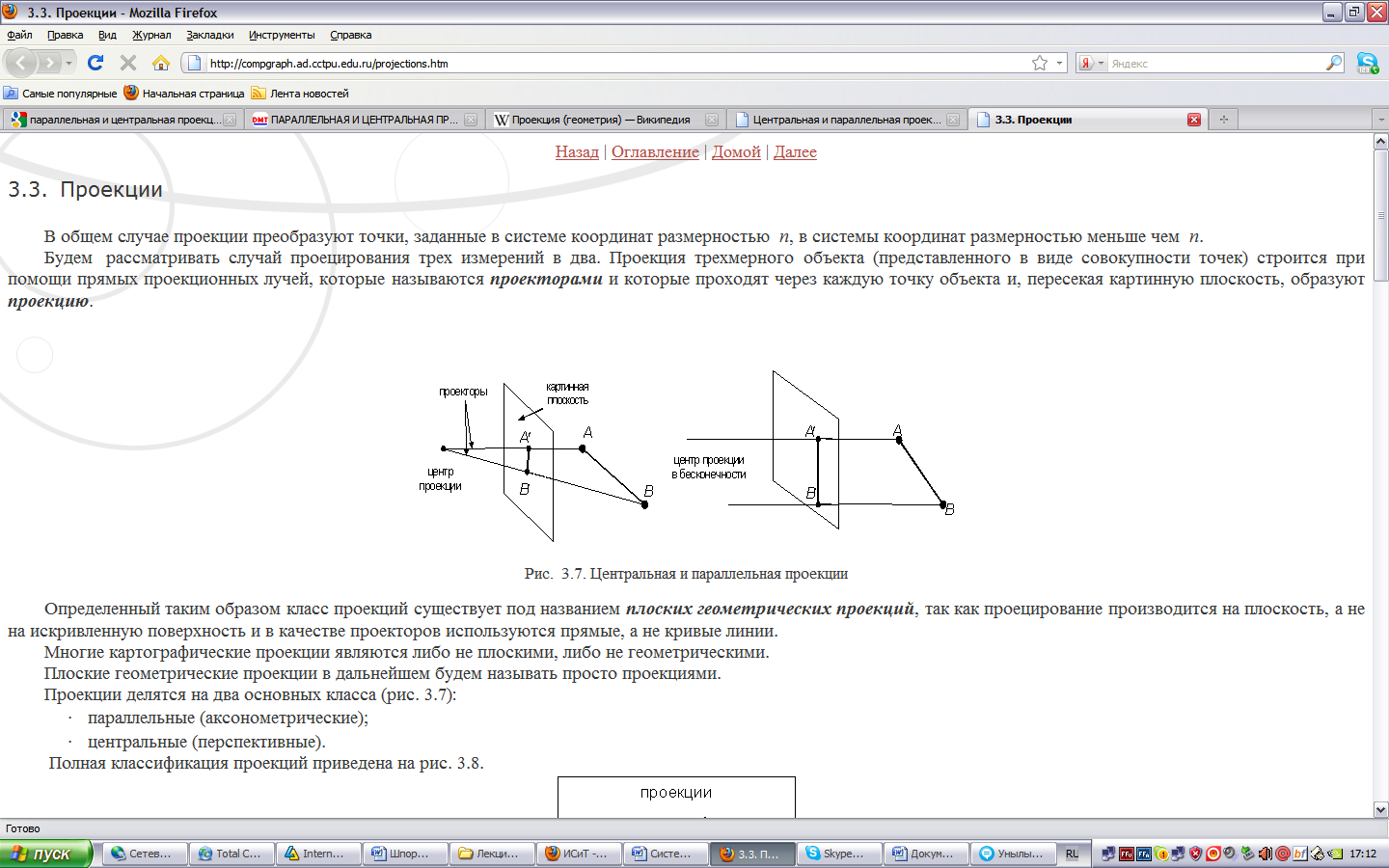
## 20. Аффинные преобразования в пространстве. Преобразования объектов: смещение, растяжение-сжатие, повороты вокруг осей координат



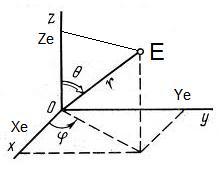


## 21. Основные типы проекций. Видовая система координат (вывод матрицы преобразования)

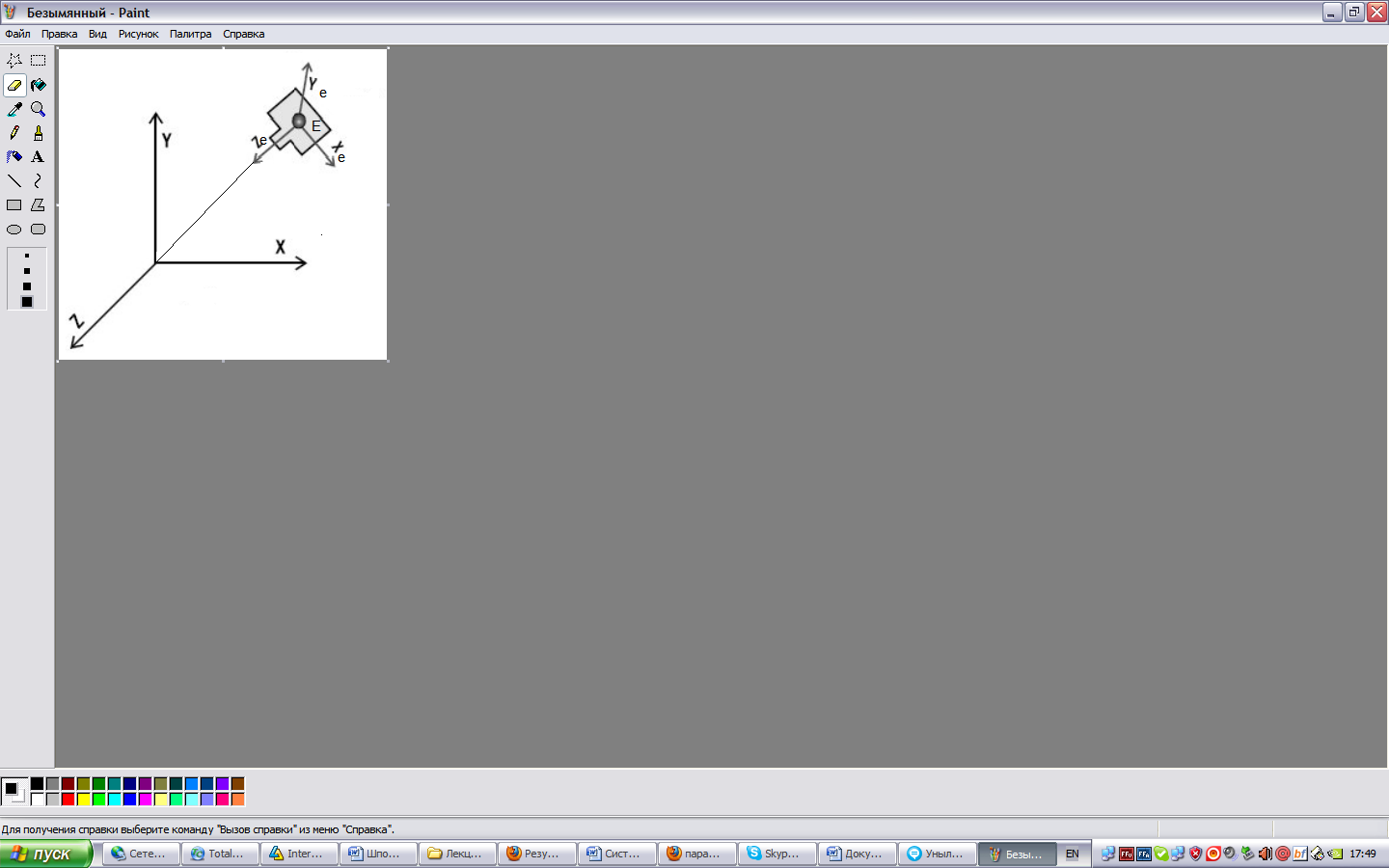
Картинная плоскость-плоскость, на которой выполняется проекция объекта. В компьютерной графике наиболее распространены центральные (рис. слева) и параллельные (рис. справа) проекции.

****

Видовая система координат

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

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

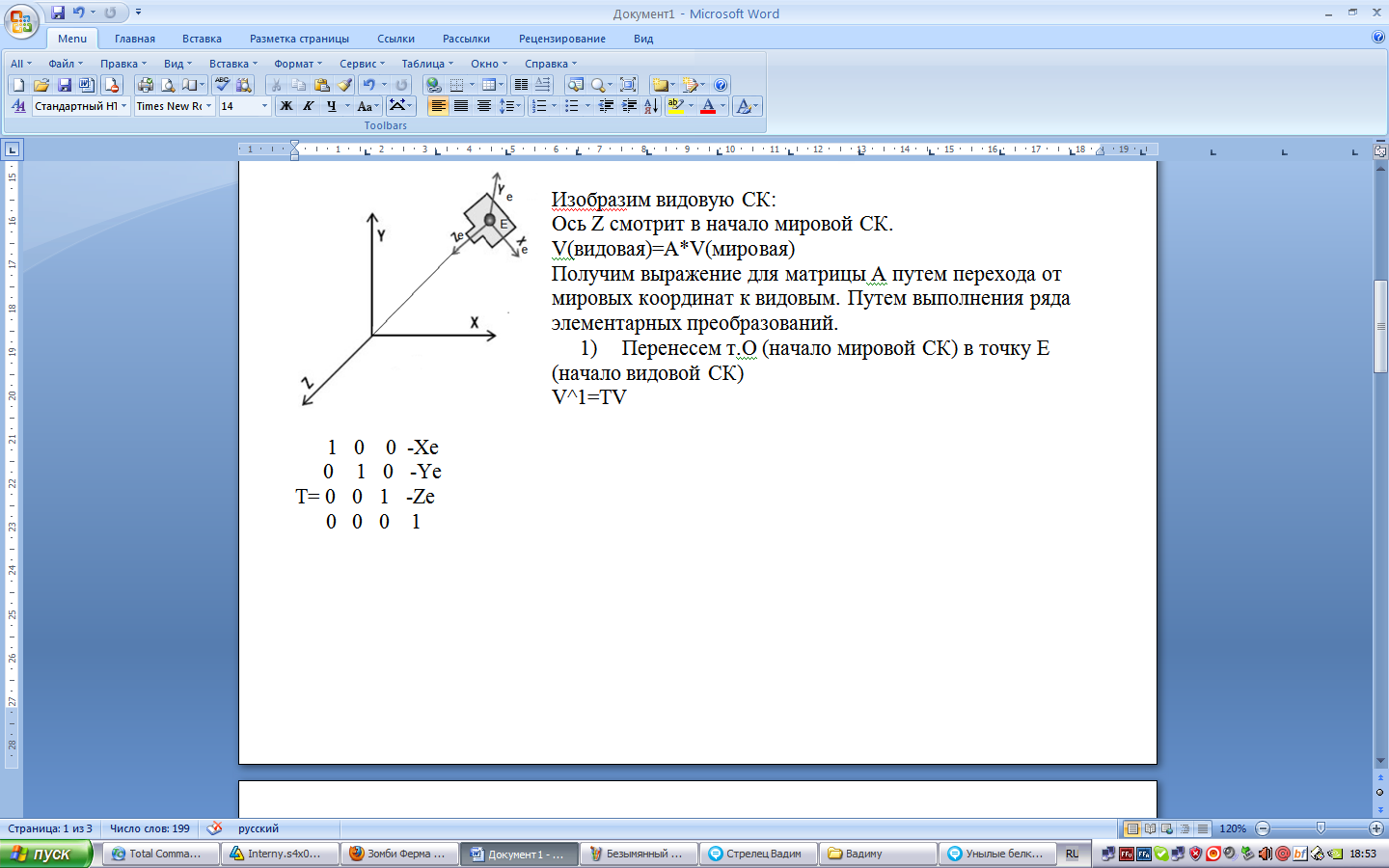
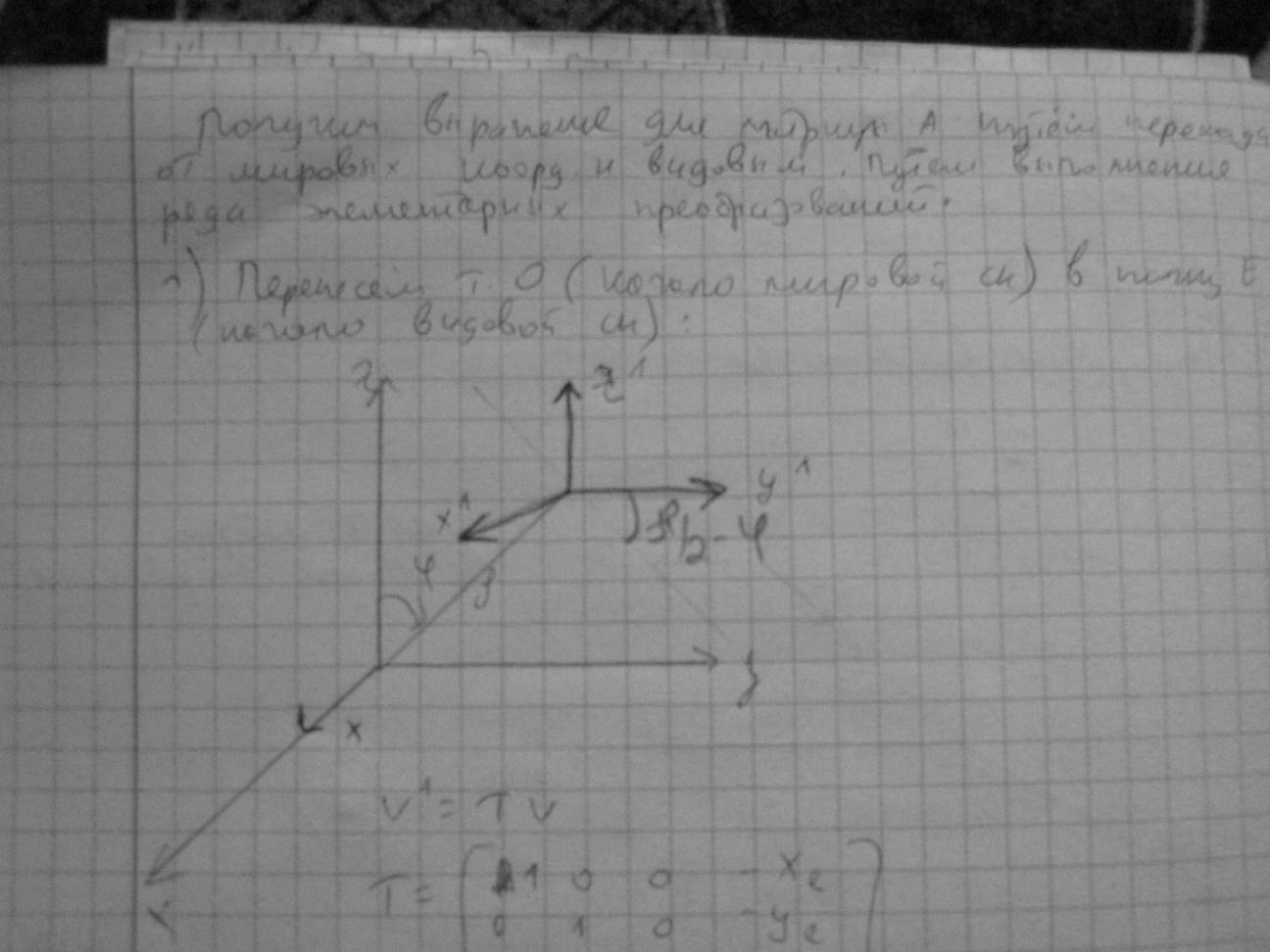
Изобразим видовую СК:

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

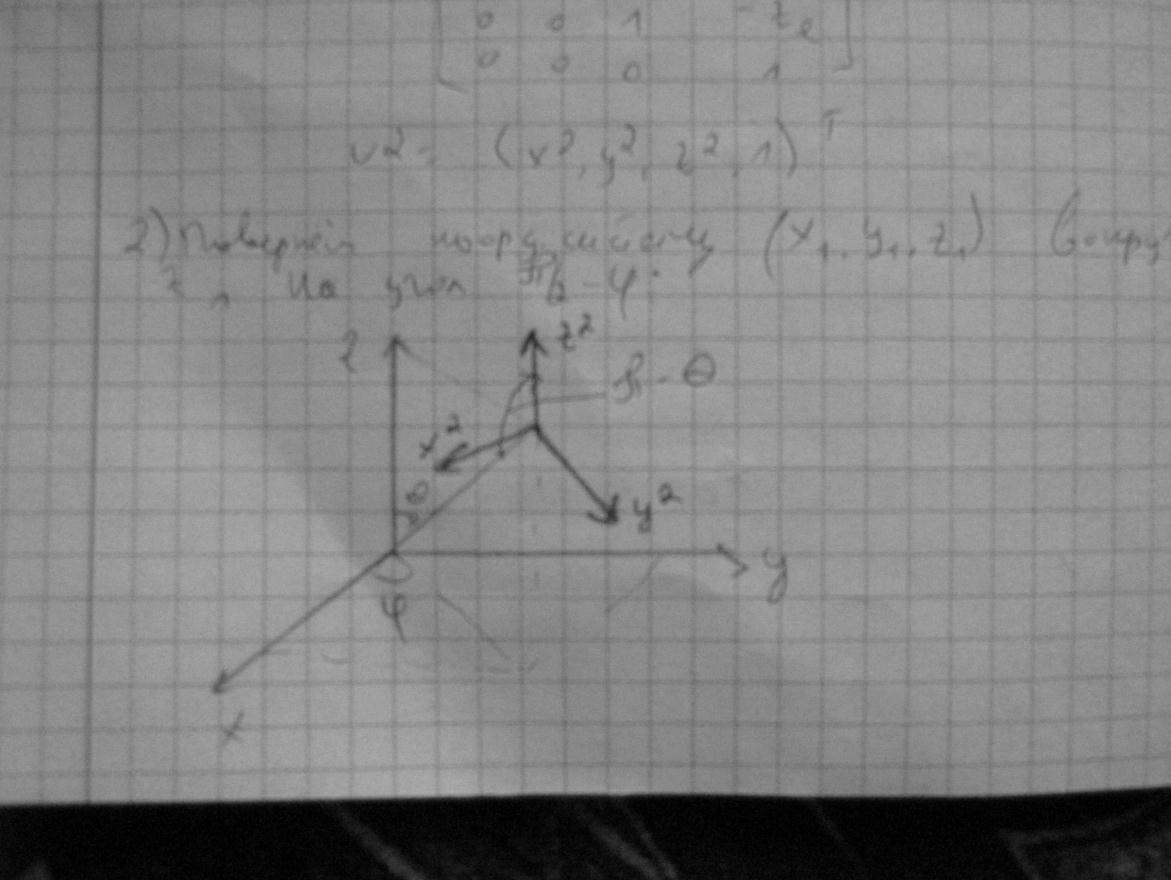
Получим выражение для матрицы А путем перехода от мировых координат к видовым. Путем выполнения ряда элементарных преобразований.

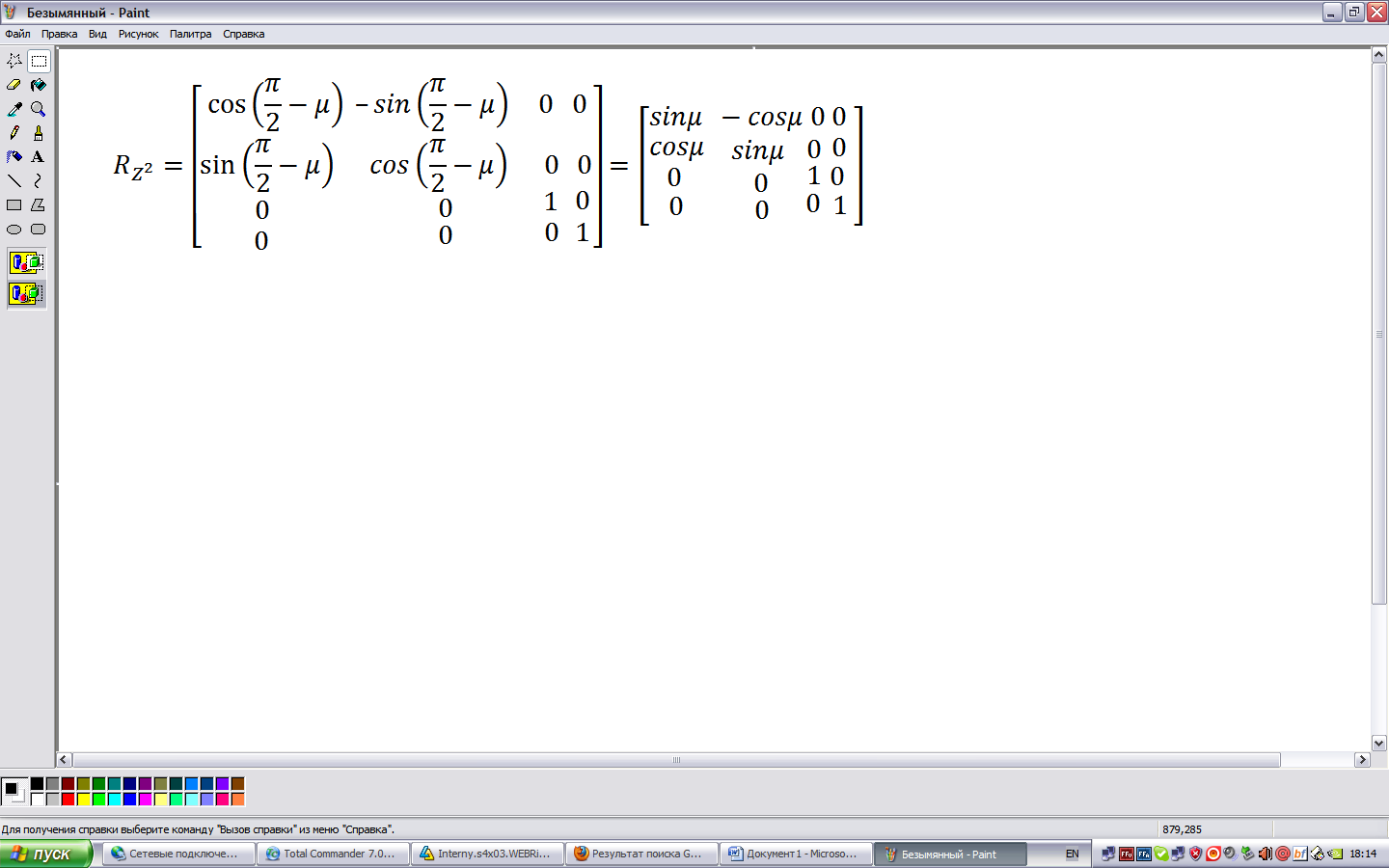
Перенесем точку О (начало мировой СК) в точку Е (начало видовой СК)

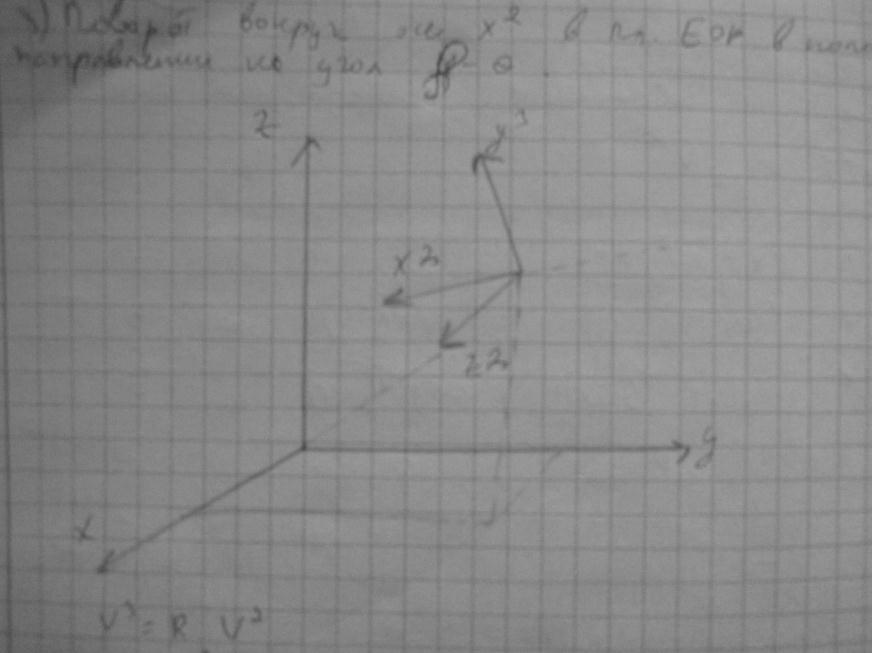
V^1=TV

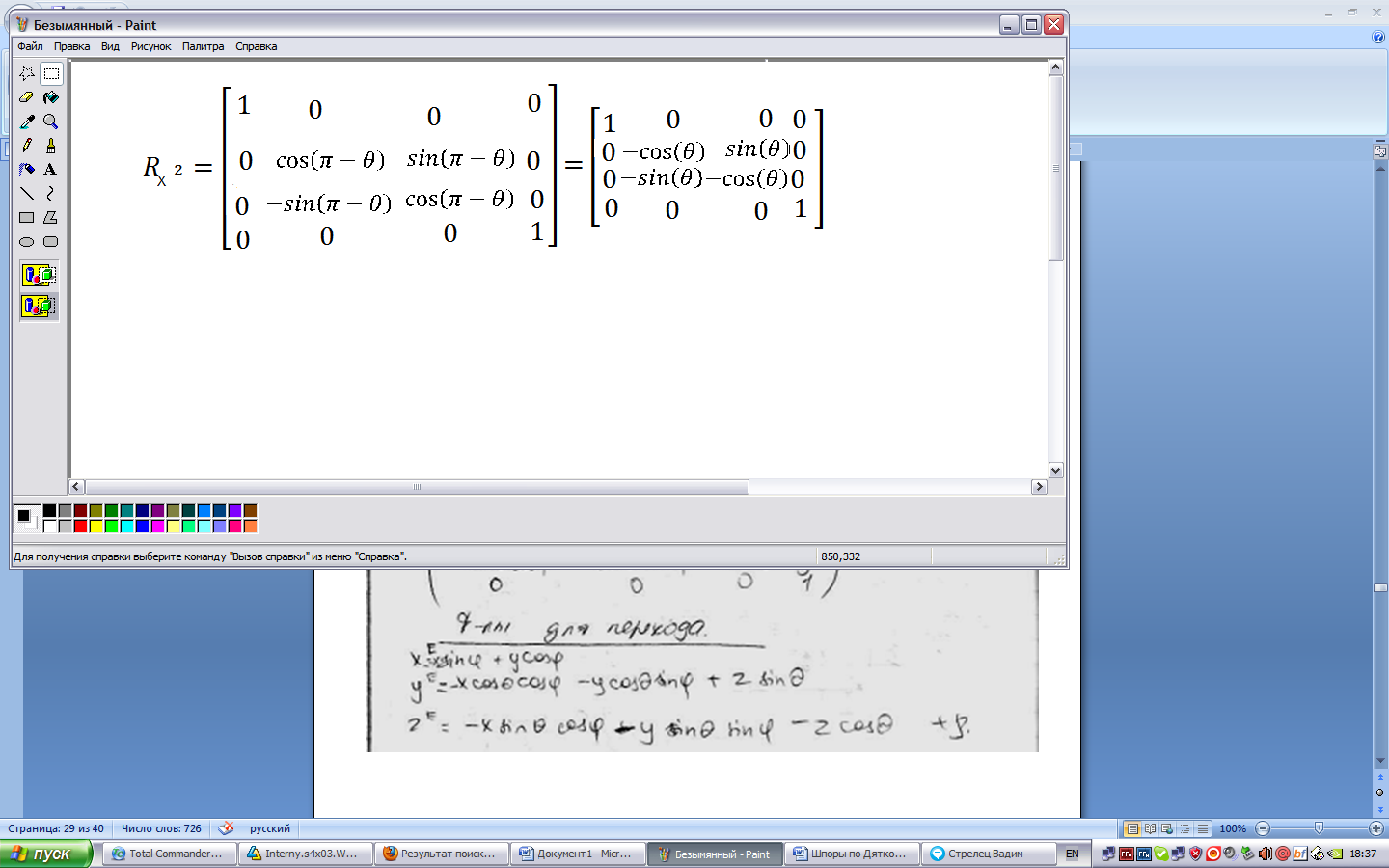


Повернем координатную систему (X1,Y1,Z1) вокруг оси Z1 на угол π/2-µ. Поворот выполняем в отрицательном направлении (против часовой стрелки) => матрица данного преобразования будет совпадать с преобразованием точки.

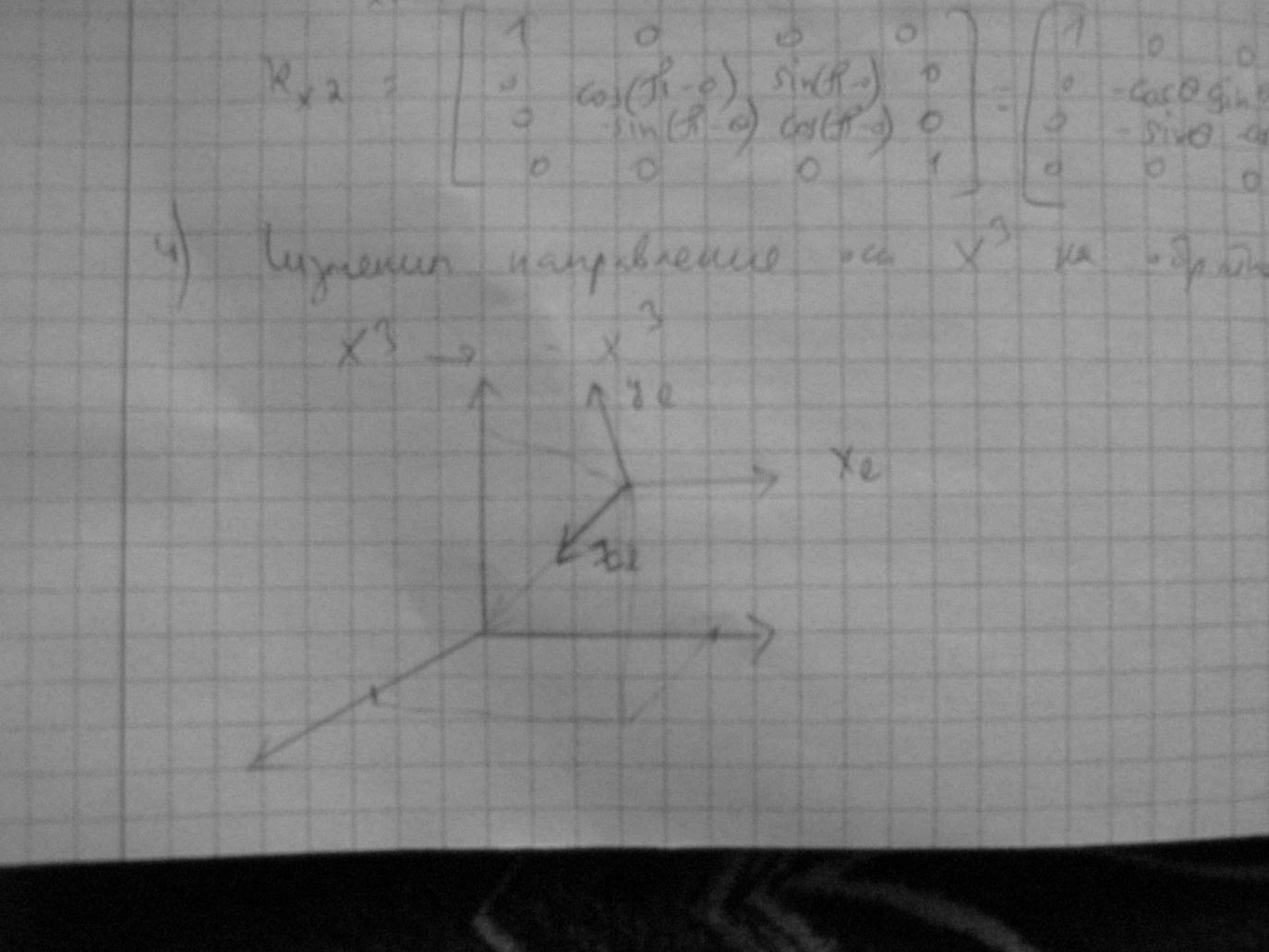
 ;

**

Поворот вокруг оси в положительном направлении на угол .

**

Изменим направление оси на обратное (-).



Вместо V3 подставим V2. Вместо V2 подставим V1

; ;

С учетом матрицы A:

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

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























Картинная плоскость – плоскость проецирования в видовой системе координат.

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

– координаты точки схода лучей.

При такой расстановке координат

– z координата объекта в видовой системе координат.

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

Из подобия треугольников и получаем пропорцию:

*;* таккак *, ;*

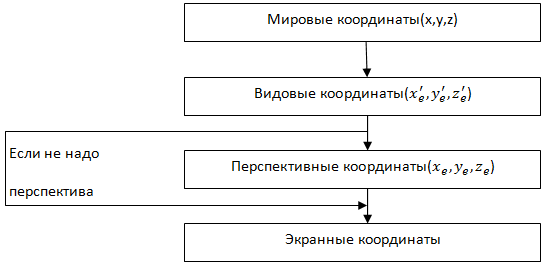
Аналогично их подобия треугольников и

Обозначим , тогда:

; ;

;

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



## 23. Модели описания поверхностей

Аналитическая модель

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

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

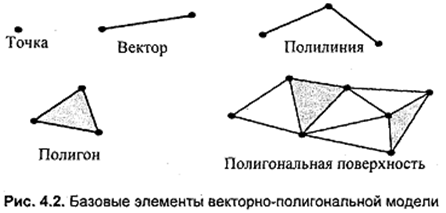


где s и t — параметры, которые изменяются в определенном диапазоне, функции *Fx*, *Fy* и *Fz* будут определять форму поверхности.

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

Векторная полигональная модель

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



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

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

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

**Отсечение нелицевых граней**

Рассмотрим многогранник, для каждой грани которого задан вектор внешней нормали (рис. 12.5). Несложно заметить, что если вектор нормали грани  () составляет с вектором , задающим направление проектирования, тупой угол, то эта грань заведомо не может быть видна. Такие грани называются ***не лицевыми***. В случае, когда соответствующий угол является острым, грань называется ***лицевой***.

|  |  |
| --- | --- |
|  |  |
| Рис. 12.5 | Рис. 12.6 |

Пусть  – угол между вектором нормали  к некоторой грани и направлением проектирования , которые заданы в мировой системе координат.

Так как для случая ***параллельного*** проектирования направление проектирования не зависит от грани (значение вектора  не зависит от номера грани ), то условие отбора ***лицевой*** грани с номером  можно записать в виде

, (12.1)

что говорит о том, что угол  является острым и его значение принадлежит отрезку . Поскольку знаменатель в (12.1) всегда положителен, то условие (12.1) можно заменить на более простое

 (12.2)

Рассмотрим случай ***центрального*** проектирования (рис. 12.6). Пусть для этого случая в видовой системе координат заданы:  – произвольная точка на некоторой грани с номером ,  – вектор внешней нормали к этой грани,  – точка, определяющая центр проектирования,  – направление проектирования для точки ,  – угол между вектором нормали  и направлением проектирования . В отличие от случая параллельного проектирования значение вектора  зависит от положения точки  на выбранной грани.

 (12.3)

Теперь можно записать условие отбора ***лицевой*** грани

 (12.4)

или более просто

 () (12.5)

Знак этого скалярного произведения в (12.5) не зависит от выбора точки на грани, а определяется тем, в каком полупространстве относительно плоскости, содержащей данную грань, лежит центр проектирования. Поэтому, определения того, является заданная грань лицевой или нет, достаточно взять произвольную точку на этой грани и проверить выполнение условия (12.5)

В случае, когда фигура представляет собой один выпуклый многогранник, удаление не лицевых граней полностью решает задачу удаления невидимых граней.

В общем случае предложенный подход хотя и не решает задачу полностью, но позволяет примерно вдвое сократить количество рассматриваемых граней [].

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

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 каждого объекта производится проверка, не закрыто ли оно гранями 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). Если ребро находится перед проверяемой гранью, из массива FACE-TABLE выбирается следующая грань и ребро сравнивается уже с ней. Если ребро оказывается за гранью, или проходит ее насквозь, приходится выполнять дополнительное действие.

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

**26. Закрашивание поверхностей. Модели отражения света**

**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 [\_\_].

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. 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  [].

, (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 (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 :

 (13.6)

где– расстояние от центра проекции до поверхности,  константа.

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

В данном разделе рассматривается задача вычисления вектора нормали к поверхности, которая возникает во многих задачах компьютерной графики.

Сначала вычислим нормаль к плоскости, которая определяется расположенными на ней тремя точками (рис. 7.6).

|  |  |
| --- | --- |
|  |  |
| Рис. 7.6 | Рис. 7.7 |

Пусть точки , и  лежат в одной плоскости.

Образуем вектора

 и .

Тогда вектор нормали  к плоскости можно вычислить как векторное произведение векторов  и 

 (7.44)

Выражение (44) используется для вычисления координат вектора нормали в случае описания поверхности векторно – полигональной моделью (см. ниже).

**Отражение света:**

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 света или 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)

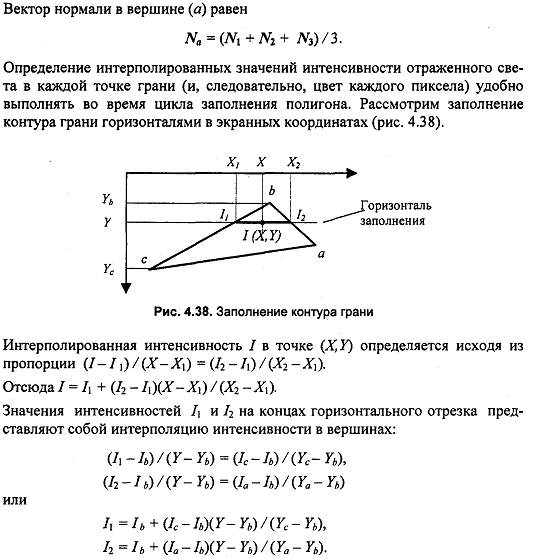
 И окончательно 

## 28. Метод закрашивания Гуро

Этот метод предназначен для создания иллюзии гладкой криволинейной поверхности, описанной в виде многогранников или полигональной сетки с плоскими гранями. Если каждая плоская грань имеет один постоянный цвет, определенный с учетом отражения, то различные цвета соседних граней очень заметны, и поверхность выглядит именно как многогранник. 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.



## 29. Графическая библиотека OpenGL. Особенности, основные возможности, архитектура, синтаксис команд

## OpenGL является одним из самых популярных прикладных программных интерфейсов (API – Application Programming Interface) для разработки приложений в области двумерной и трехмерной графики.

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. 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.

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

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.

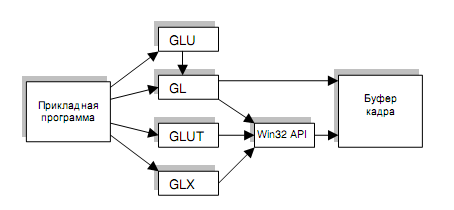
Функции описания источников света служат для описания положения и параметров источников света, расположенных в трехмерной сцене.

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.

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.

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.

Интерфейс:



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

type **glCommand\_name[1 2 3 4][b s i f d ub us ui][v]**(type1 arg1,…,typeN argN)

## 30. Графическая библиотека OpenGL. Структура GLUT-приложения. Пример

Инициализация проводится с помощью функции:

void glutInit (int \*argcp, char \*\*argv), где переменная argcp – указатель на стандартную переменную argc описываемую в функции main(), а argv – указатель на параметры, передаваемые программе при запуске, который описывается там же.

void glutInitWindowPosition (int x, int y) – служит для установки положения окна OpenGL на экране, здесь x есть X -координата левого верхнего угла окна, y есть Y – координата левого верхнего угла окна.

void glutInitWindowSize (int width, int height) – служит для установки размеров окна на экране, здесь width – ширина окна, height – высота окна.

int glutCreateWindow ( char \* name ), где параметр name задает заголовок окна.

void glutDisplayFunc (void (\*func) (void)) – служит для установки обработчика события, связанного с необходимостью перерисовки содержимого окна.

void MyIdle(void)

{

/\*Код, который меняет переменные, определяющие следующий кадр \*/

…

};

void MyDisplay(void)

{

/\* Код OpenGL, который отображает кадр \*/

…

/\* После рисования переставляем буферы \*/

glutSwapBuffers();

};

void main(int argcp, char \*\*argv)

{

/\* Инициализация GLUT \*/

glutInit(&argcp, argv);

glutInitWindowSize(640, 480);

glutInitWindowPosition(0, 0);

/\* Открытие окна \*/

glutCreateWindow("My OpenGL Application");

/\* Выбор режима: двойной буфер и RGBA цвета \*/

glutInitDisplayMode(GLUT\_RGBA | GLUT\_DOUBLE | GLUT\_DEPTH);

/\* Регистрация вызываемых функций \*/

glutDisplayFunc(MyDisplay);

glutIdleFunc(MyIdle);

/\* Запуск механизма обработки событий \*/

glutMainLoop();

};

## 31. Графическая библиотека OpenGL. Рисование графических примитивов. Дисплейные списки. Удаление нелицевых граней. Вывод текста. Примеры

**Вершины и примитивы**

Вершина является атомарным графическим примитивом OpenGL и определяет точку, конец отрезка, угол многоугольника и т.д. Все остальные примитивы формируются с помощью задания вершин, входящих в данный примитив. Например, отрезок определяется двумя вершинами, являющимися концами отрезка. С каждой вершиной ассоциируются ее атрибуты. В число основных атрибутов входят ***положение*** вершины в пространстве, ***цвет*** вершины и ***вектор*** нормали.

***Положение вершины в пространстве***

Положение вершины определяются заданием ее координат в двух-, трех-, или четырехмерном пространстве (однородные координаты). Это реализуется с помощью нескольких вариантов команды **glVertex\***:

**void glVertex[2 3 4][s i f d] (type x, …)**

**void glVertex[2 3 4][s i f d]v (type \*v)**

**Дисплейные списки**

В традиционных языках программирования существуют функции и процедуры – т. е. можно выделить определенный набор команд, запом­нить его в некотором одном определенном месте и вызывать каждый раз, когда возникает потребность в соответствующей последовательности ко­манд. Подобная возможность существует и в OpenGL – набор команд OpcnGL можно запомнить в так называемый дисплейный список (***display list***), при этом все команды и данные переводятся в некоторое внутреннее представление, наиболее удобное для данной реализации OpenGL, и затем вызвать при помощи всего одной команды.

Каждому дисплейному списку соответствует некоторое целое число, идентифицирующее этот список.

Для создания нового дисплейного списка необходимо поместить все команды, которые должны в него войти, между следующими операторными скобками:

**void glNewList (GLuint list, GLenum mode)**,

**void glEndList ()**,где параметр **list** задает номер списка, а параметр **mode** определяет режим обработки команд: **GL\_COMPILE,** **GL\_COMPILE\_AND\_EXECUTE.**

После того, как список создан, его можно вызвать командой:

**void glCallList (GLuint list)**

Для удаления списков используется команда:

**void glDeleteLists (GLint list, GLsizei range)**

**Вывод текста:** существуют два типа шрифтов – ***штриховой (эскизный)*** и ***растровый***.

Растровый символ GLUT можно получить с помощью функции

**glutBitmapCharacter (font, character)**

Эскизный символ изображается путем вызова следующей функции.

**glutStrokeCharacter (font, character)**

## 32. Графическая библиотека OpenGL. Системы координат. Модельно-видовые преобразования. Примеры

В OpenGL используются как основные три системы координат: ***левосторонняя, правосторонняя и оконная***. Первые две системы являются трехмерными и отличаются друг от друга направлением оси **Z**: в правосторонней она направлена на наблюдателя, в левосторонней – в глубину экрана. Ось **X** направлена вправо относительно наблюдателя, ось **Y** – вверх.

Рассматривая какой-либо трехмерный объект, мы всегда определяем его по­ложение и размеры относительно некоторой привычной, и удобной в на­стоящий момент системы координат, связанной с реальным миром. Такая исходная система координат в компьютерной графике является **правосто­ронней** и называется **мировой системой координат (МСК).** В ней над объектами выполняются заданные действия, например аффинные преобразования. Для того чтобы можно было изобразить объект на экране, его необходимо предварительно перевес­ти (или преобразовать) в другую систему координат, которая связана с точ­кой наблюдения и носит название **видовой системы координат или системой координат наблюдателя (СКН).** Эта система координат является левосторонней. В ней рассчитываются цвета объектов и выполняются заданные отсечения. Далее, координаты **СКН** переводятся в ***нормированные (нормализованные)*** координаты **(НК)**, которые лежат в диапазоне **[ -1, 1 ]**. И, наконец, любое трехмерное изображение мы всегда рисуем на двумерном экране, который имеет свою **экранную (оконную) систему координат (ОСК)**.

**Модельно-видовые преобразования**

К модельно-видовым преобразованиям будем относить перенос, поворот и изменение масштаба вдоль координатных осей. Для проведения этих операций достаточно умножить на соответствующую матрицу каждую вершину объекта и получить измененные координаты этой вершины:



где M – матрица модельно-видового преобразования. Перспективное преобразование и проектирование производится аналогично. Сама матрица может быть ***создана*** с помощью следующих команд:

**void glTranslate[f d] (GLtype x, GLtype y, GLtype z)**,

**void glRotate[f d] (GLtype angle, GLtype x, GLtype y, GLtype z)** ,

**void glScale[f d] (GLtype x, GLtype y, GLtype z)**.

**33. Графическая библиотека OpenGL. Проекции. Область вывода. Создание анимации. Примеры**

OpenGL поддерживает два вида прекций – ***ортографическую (параллельную)*** и ***перспективную*** проекции.

В OpenGL существуют стандартные команды для задания ортографической (параллельной) и перспективной проекций, выполнение которых ***устанавливает в системе координат наблюдателя область видимости графических данных***.

Первый тип проекции может быть задан командами:

**void glOrtho (GLdouble x\_left, GLdouble x\_right, GLdouble y\_bottom, GLdouble y\_top, GLdouble z\_near, GLdouble z\_far)**

**void gluOrtho2D (GLdouble x\_left, GLdouble x\_right, GLdouble y\_bottom, GLdouble y\_top)**

При использовании ***ортографической проекции*** в качестве области видимости необходимо определить ***усеченный объем видимости (параллелепипед видимости)***.

При использовании ***перспективной проекции*** в качестве области видимости необходимо определить не параллелепипед, а ***усеченную пирамиду***. Для этого в OpenGL реализованы две команды:

**void glFrustrum(GLdouble left, GLdouble right, GLdouble top, GLdouble bottom, GLdouble znear, GLdouble zfar)**

**void gluPerspective (GLdouble angley, GLdouble aspect, GLdouble znear, GLdouble zfar).**

***Область вывода*** представляет собой прямоугольник в оконной системе координат, размеры которого задаются командой:

**void glViewPort (GLint x, GLint y, GLint width, GLint height)**

Значения всех параметров задаются в ***пикселах***. Параметры **x** и **y** определяют координаты ***левого нижнего угла*** прямоугольника ***вывода в оконной системе координат*** (видовой порт)и ***по умолчанию принимаются равными нулю***. Размеры оконной системы координат определяются текущими размерами окна приложения, точка **(0,0)** находится в ***левом нижнем углу*** окна.

**Создание анимации.**

Для создания анимации в OpenGL необходимо выпонить следующие действия.

1. Установить режим двойной буферизации

glutInitDisplayMode(GLUT\_RGB|**GLUT\_DOUBLE**)

1. Зарегистрировать и создать функцию обратного вызова (например функцию с именем IdleFunction), которая вызывается системой когда нет других событий.

Внутри функции IdleFunction описываются действия, которые влияют на вид изображения после перерисовки окна. Например, изменяется угол поворота объекта. glutPostRedisplay( ); // Обновить окно. В функции рисования OnDraw( ) использовать команду переключения буферов – glutSwapBuffers().

Для управления процессом запуска и остановки анимации можно использовать:

glutIdleFunc(IdleFunction) – разрешает выполнение функции IdleFunction;

glutIdleFunc(NULL) – блокирует выполнение функции IdleFunction.

34. Графическая библиотека OpenGL. Материалы и освещение. Примеры

Для создания реалистичных изображений необходимо определить как свойства самого объекта, так и свойства среды, в которой он находится. Первая группа свойств включает в себя параметры материала, из которого сделан объект, способы нанесения текстуры на его поверхность, степень прозрачности объекта. Ко второй группе можно отнести количество и свойства источников света, уровень прозрачности среды, а также модель освещения. Все эти свойства можно задавать, вызывая соответствующие команды OpenGL.

**Модель освещения:**

В OpenGL используется модель освещения, в соответствии с которой цвет точки определяется несколькими факторами: ***свойствами материала и текстуры***, ***величиной нормали в этой точке***, а также ***положением источника света и наблюдателя***. Для корректного расчета освещенности в точке надо использовать единичные нормали, однако команды типа **glScale\*()**, могут изменять длину нормалей. Чтобы это учитывать, необходимо использовать режим нормализации векторов нормалей, который включается вызовом команды **glEnable(GL\_NORMALIZE)**.

Для задания глобальных параметров освещения используются команды

**void glLightModel[i f] (GLenum pname, GLenum param)**,

**void glLightModel[i f]v (GLenum pname, const GLtype \*params)**.

Спецификация материалов:

Для задания параметров текущего материала используются команды

**void glMaterial[i f] (GLenum face, GLenum pname, GLtype param),**

**void glMaterial[i f]v (GLenum face, GLenum pname, GLtype \*params)**

С их помощью можно определить ***рассеянный***, ***диффузный*** и ***зеркальный***  цвета материала, а также степень зеркального отражения и интенсивность излучения света, если объект должен светиться.

**Описание источников света:**

Определение свойств материала объекта имеет смысл, только если в сцене есть источники света. Иначе все объекты будут черными. Добавить в сцену источник света можно с помощью команд

**void glLight[i f] (GLenum light, GLenum pname, GLfloat param)**,

**void glLight[i f]v (GLenum light, GLenum pname, GLfloat \*params)**

Параметр **light** однозначно определяет источник света. Он выбирается из набора специальных символических имен вида **GL\_LIGHTi** , где **i** должно лежать в диапазоне от **0** до константы **GL\_MAX\_LIGHT**, которая ***обычно не превосходит восьми***.

Для использования освещения сначала надо установить соответствующий режим вызовом команды **glEnable(GL\_LIGHTNING)**, а затем включить нужный источник командой **glEnable(GL\_LIGHTi)**. Еще раз обратим внимание на то, что при ***выключенном*** освещении цвет вершины равен текущему цвету, который задается командами **glColor\*()**. При ***включенном*** освещении цвет вершины вычисляется исходя из информации о материале, нормалях и источниках света.

35. Графическая библиотека OpenGL. Структура приложения с использованием библиотеки MFC

Если OpenGL используется в MFC, то необходимо заполнить структуру PIXELFORMATDESCRIPTOR и вызвать SetPixelFormat(). В структуре необходимо указать PFD\_SUPPORT\_OPENGL | PFD\_DOUBLEBUFFER (формат с поддержкой OpenGL и двойной буферизацией).

int CChildView::OnCreate(LPCREATESTRUCT lpCreateStruct)

{

if (CWnd::OnCreate(lpCreateStruct) == -1)

return -1;

PIXELFORMATDESCRIPTOR pfd = // Описатель формата

{

sizeof(PIXELFORMATDESCRIPTOR),// Размер структуры

1, // Номер версии

PFD\_DRAW\_TO\_WINDOW | // Поддержка GDI

PFD\_SUPPORT\_OPENGL | // Поддержка OpenGL

PFD\_DOUBLEBUFFER, // Двойная буферизация

PFD\_TYPE\_RGBA, // Формат RGBA, не палитра

24, // Количество плоскостей

// в каждом буфере цвета

24, 0, // Для компоненты Red

24, 0, // Для компоненты Green

24, 0, // Для компоненты Blue

24, 0, // Для компоненты Alpha

0, // Количество плоскостей буфера Accumulation

0, // То же для компоненты Red

0, // для компоненты Green

0, // для компоненты Blue

0, // для компоненты Alpha

32, // Глубина Z-буфера

0, // Глубина буфера Stencil

0, // Глубина буфера Auxiliary

0, // Теперь игнорируется

0, // Количество плоскостей

0, // Теперь игнорируется

0, // Цвет прозрачной маски

0 // Теперь игнорируется

};

if ( (pixelformat = ChoosePixelFormat(m\_pDC->GetSafeHdc(), &pfd)) == 0 )

{

MessageBox("ChoosePixelFormat failed");

return FALSE;

}

if (SetPixelFormat(m\_pDC->GetSafeHdc(), pixelformat, &pfd) == FALSE)

{

MessageBox("SetPixelFormat failed");

return FALSE;

}

}