ФГБОУ ВПО МГТУ «СТАНКИН»

Кафедра инженерной графики

КОМПЬЮТЕРНАЯ ГЕОМЕТРИЯ И ГРАФИКА

Учебник

Москва
2019

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Учебник является базовой подборкой информации в области геометрии, реализующей алгоритмы компьютерного моделирования графических технологий. Представляет собой материал, необходимый для изучения основ компьютерной геометрии, широко применяемой в программировании компьютерной графики. В книге собраны и доработаны материалы существующих учебных пособий, учебников и монографий, которые, по мнению автора наиболее удачно отражают суть решаемых задач компьютерной графики. К таким работам относятся ставшие классикой монографии *Л. Аммерала, Дж. Роджерса*, а также учебные пособия современных российских авторов [1-6].

Учебник содержит два основных раздела: компьютерная геометрия и локальная компьютерная геометрия. В первом разделе собраны все основные математические аспекты, позволяющие реализовывать алгоритмы компьютерной графики с применением координатных геометрических 2D и 3D дискретных моделей. Во втором разделе рассматривается наработанный математический аппарат, реализующий те же алгоритмы применительно к моделям, описанным локальными геометрическими характеристиками.

Каждый из разделов разбит на темы, позволяющие последовательно увязать знакомство принципов программирования интерактивной компьютерной графики с построением простых и сложных геометрических объектов, и различных способов их представления в графических приложениях.

Курс предусматривает приобретение базовых навыков программирования на языке C++, достаточных для построения простых Windows-приложений, но позволяющих решать основные задачи современной компьютерной графики с применением традиционного интерактивного графического посредника − «мышь».

Материал учебного пособия не требует предварительного глубокого изучения языка C++ и допускает общие образовательные навыки в

программировании, т.е. знакомство с такими базовыми понятиями как операция, операнд, цикл, массив, тип данных, подключение библиотек и т.п., с чем, как правило, должны знакомить на школьном предмете информатики в примерах изучения таких языков как Basic, Pascal, Python и т.п.

Автор умышленно выбрал за основу средства программирования Windows API (*программный интерфейс приложений – Application Program Interface*), позволяющие осуществлять построение Windows-приложений на C++ без привязки к конкретной среде программирования. Несмотря на то, что примеры рассмотрены для среды Visual C++ 2010 текст программы написан таким образом, что проект может создаваться в любой среде, поддерживающей язык C++. При этом текст программы, предложенный учебником, не претерпевает особых изменений.

Особое внимание и умышленно с дополнительными подробностями уделяется рассмотрению математических аспектов компьютерной графики, поскольку курс рассчитан читателя с начальной подготовкой основ высшей математики и допускает параллельное изложение этих предметов.

В пособии рассмотрены такие разделы высшей математики как: матричное преобразование, аппроксимация функции, интерполяция функции, интегральное и дифференциальное исчисления и т.п. с попыткой предать им геометрический, а также графической смысл.

Стоит сразу отметить, что учебное пособие не предусматривает изучение и применение существующих графических библиотек OpenGL или DirectX. Оно направлено на знакомство с принципами построения некоторых основных графических функций, содержащихся в этих библиотеках. Изучение вышеуказанных библиотек предоставляется более позднему и более профессиональному циклу изучения программирования компьютерной графики. При этом знания, предоставленные в этом учебнике, позволят достигнуть боле глубокого теоретического понимания математических принципов построения современных средств компьютерной графики.

1. ИСТОРИЯ РАЗВИТИЯ КОМПЬЮТЕРНОЙ ГРАФИКИ.

Принято начало истории компьютерной графики относить к появлению первых разработок графических устройств, открывших новую изобразительную эру компьютерного приложения. Трудно не согласиться с общим мнением, хотя следует отметить, что уже с применением алфавитно-цифровых печатающих устройств (40-50 гг.) появились первые компьютерные изображения, несущие достаточно графическое представление. В качестве единицы изображения рассматривался образ символа. В этом случае, плотность изображения символа представляла интенсивность в изображаемой «единице» (Например, в славянском шрифте наименьшая плотность обозначалась точкой «.», а наибольшая буквой «**Ж**» или «**М**»). Набор символов образует строку, а набор строк – «символьный растр».

Рис.1.1. Пример символьной графики «Джоконда»

По сути, этот принцип впоследствии переходит от символьной в растровую графику. Требуется, к примеру, изобразить график функции разложим растр по строкам и столбцам. Незамысловатый алгоритм распределит необходимое пространственное положение символов в метрике символьного растра. Таким образом создавались и более сложные изображения. Созданная печатающими устройствами «Джоконда» (рис.1.1) или распятый «Иисус» в обязательном порядке украшали комнаты программистов того времени.

Первой машиной, где ЭЛТ (электронно-лучевая трубка) использовалась в качестве вывода, считается ЭВМ *Whirlwind-I ((h)wərl˩wind)* (Ураган-I),

изготовленная в 1950 г. в Массачусетском технологическом институте. Это был дисплей с произвольным сканированием луча для вывода изображения.

Как отметил один из разработчиков *WHIRLWIND* Норм Тейлор, компьютер «содержал около четверти акра электроники (примерно 1000 м^2) и имел дисплей». Комментатор *Эдвард Мирроу*, в 1951 году провел первое «интервью» с компьютером в телевизионной программе. Тейлор заметил тогда: «Было ясно, что дисплеи привлекают внимание потенциальных пользователей, а машинное кодирование – нет».

WHIRLWIND стал основой создания опытного образца командно-управляемой системы воздушной защиты, разработанной как средство преобразования данных, полученных от радара, в наглядную форму (Рис.1.2).

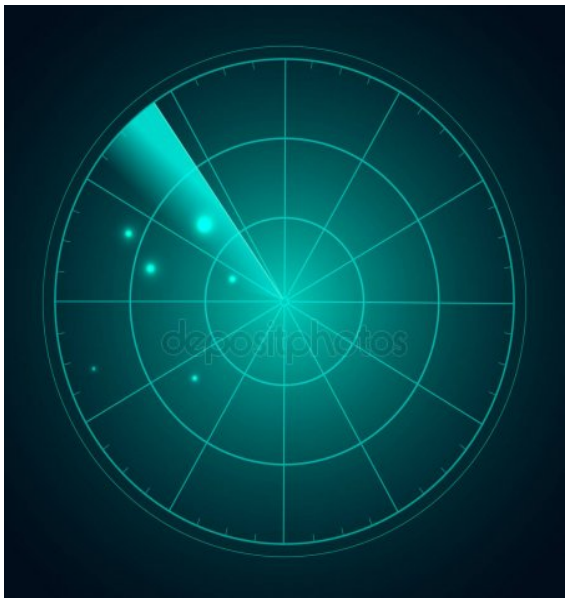


Рис.1.2. Визуализация данных радара.

С этого эксперимента начинается этап развития ***векторных дисплеев***.

Вектором называется отрезок направленной прямой. Отсюда и происходит название *векторный дисплей*. При перемещении луча по экрану в точке, на которую попал луч, возбуждается свечение люминофора экрана. Это свечение достаточно быстро прекращается при перемещении луча в другую позицию (обычное время послесвечения менее 0,1 сек.). Поэтому, для того чтобы изображение было постоянно видимым, приходится его перевыдавать (регенерировать изображение). Необходимость перевыдачи изображения

требует сохранения его описания в специально выделенной памяти, называемой *памятью регенерации*. Само описание изображения называется *дисплейным файлом*. Понятно, что такой дисплей требует достаточно быстрого процессора для обработки дисплейного файла и управления перемещением луча по экрану.

К середине 1960-х наступил период плодотворной работы и в промышленных приложениях компьютерной графики. Под руководством *Тирбера Мофетта* и *Нормана Тейлора* фирма Itek разработала цифровую электронную чертежную машину. В 1964 году *General Motors* представила свою DAC-1 - систему автоматизированного проектирования, разработанную совместно с IBM. К октябрю 1966 года даже *Wall Street Journal* уже публиковал статьи по проблемам компьютерной графики.

1.1. **Полная интерактивная графическая система**

Полная интерактивная графическая система состоит из четырех основных подсистем: ЭВМ, дисплейный процессор (ДП), устройство отображения (дисплей) и диалоговые устройства. К ЭВМ подключаются два устройства для получения твердых копий — печатающее устройство и графопостроитель (рис.1.1.1). В качестве устройства отображения в интерактивной графике обычно использовалась электронно-лучевая трубка (ЭЛТ). Существует два основных типа ЭЛТ: без запоминания изображения и с запоминанием изображения. В первом случае для поддержания на экране устойчивого изображения необходимо обновлять его, т.е. регенерировать от 30 до 60 раз в секунду по цифровому представлению, хранимому в буфере. ЭЛТ с запоминанием изображения хранит его в виде внутреннего распределения зарядов на экране и для нее не нужен ни буфер для цифрового представления, ни цикл регенерации.

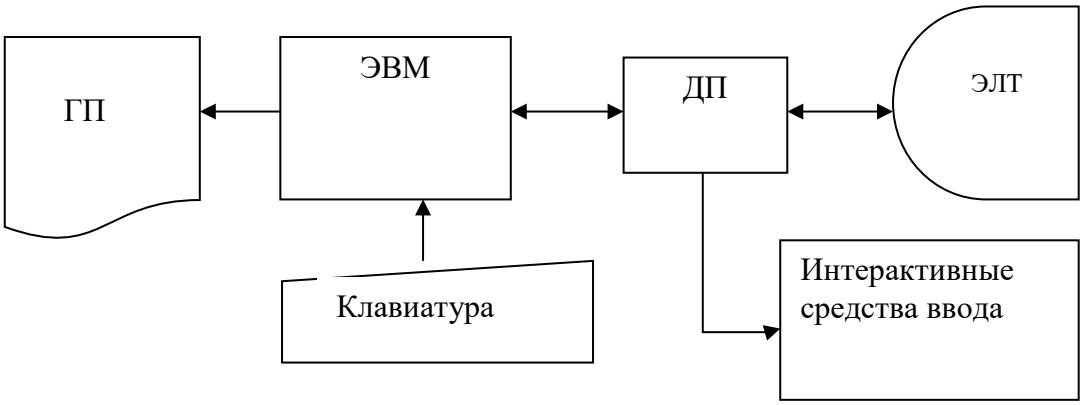


Рис.1.1.1. Общая схема технологических блоков интерактивной графической системы

Дисплейный процессор можно рассматривать как специализированный процессор, имеющий собственный набор команд, особые форматы данных и свой счетчик команд. Этот процессор выполняет последовательность дисплейных команд (дисплейная программа или ДП-программа) и порождает рисунок на *устройстве изображения.*

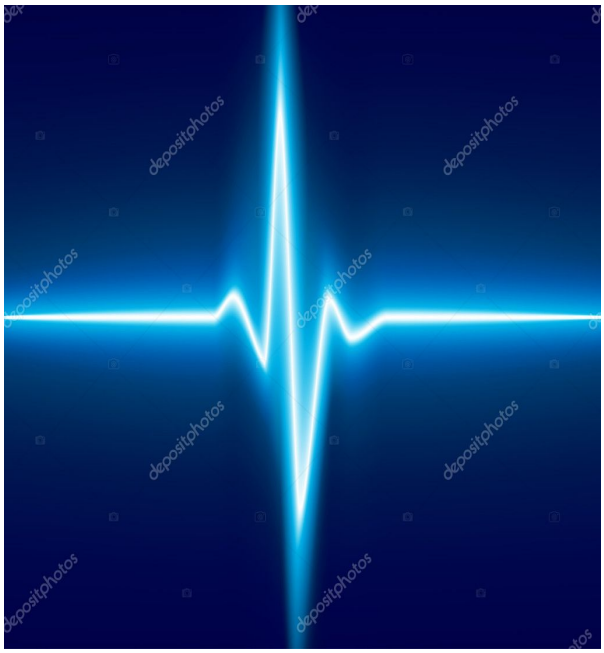


Рис.1.1.2. Векторное изображение

ДП может создавать изображение путем произвольного сканирования или путем растрового сканирования.

В системе с произвольным сканированием части рисунка могут появляться на дисплее в любом порядке. Например, линию кардиограммы (рис.1.1.2) можно изобразить путем переноса (отклонения) начальную точку середины экрана, включения электронного луча и, последовательного его смещения вдоль линий, образующих контур кардиограммы, от одной конечной точки до другой.

В системе с *растровым (телевизионным) сканированием* рисунок разбивается на горизонтальные строки. Сначала появляются все части рисунка, находящиеся в первой строке (слева направо), затем все части рисунка, принадлежащие второй строке и т.д.

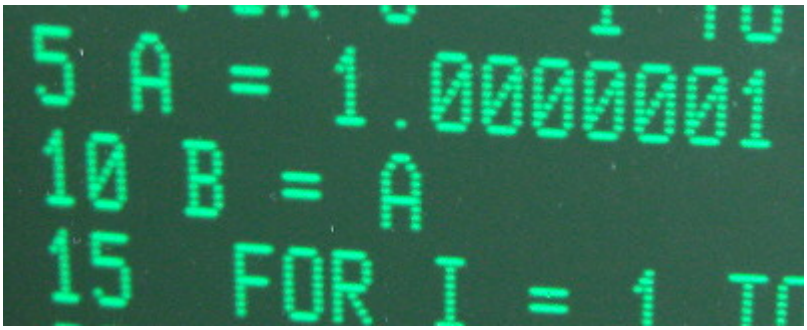


Рис.1.1.3. Растровое изображение

Во время перемещения луча слева направо, его интенсивность модулируется для создания различных уровней тона. При достижении правого края растра луч выключается, переводится к левому краю, смещаясь при этом на одну единицу вниз, и снова включается. После того как будут отработаны все строки растра, луч возвращается в левый верхний угол. Большинство растровых графических систем использует от 256 до 1024 строк. Чем больше строк, тем выше качество изображения уровней тона.

Растровое сканирование луча. Прогресс в технологии микроэлектроники привел к тому, что с середины 70-х годов подавляющее распространение получили дисплеи с растровым сканированием луча.

Первые персональные компьютеры (ПК) появились в результате не только перехода на новую элементную базу ЭВМ, но и развитию растровой технологии, позволившей обеспечить комфортный диалог с компьютером в системе ввода-вывода.

Первоначально, растровая технология опиралась на работу в текстовом режиме, где растр чётко делился на строки и расстояние между ними. Строки заполняли матрицы символов, и графические изображения представляли собой совокупность специализированных символов, стыкующихся в различных комбинациях. Для построения более сложных изображений

требовалось писать графические программы, инициализирующие графический режим, обеспечивая доступ к значению отдельного пикселя в памяти, отведённой под изображение. Существовавшая на тот период дисковая операционная система DOS работала в текстовом режиме. Не было и речи о системных графических редакторах (типа *MS Office*), и текстовые редакторы редко обзаводились возможностью ввода и редакции математических формул. Например, Chairwriter – один из популярных текстовых редакторов 80-х годов имел возможность ввода и редактирования математических формул в подстроковом и надстроковом текстовых режимах и, по понятным причинам, отсутствовала функция вставки изображения. По сути, формула конструировалась из множества спецсимволов. Но это уже было достаточно удобно для написания научных работ. Рисунки при этом выводились на отдельных страницах.

Диалог с компьютером обеспечивался в директивном режиме, что требовало от пользователя заучивания многих конструкций команд и ключей.

Вначале 90-х на отечественном рынке появляются первые графические персональные системы фирмы *Macintosh* (прообраз системы *Windows*). Эти операционные системы и обслуживающее ПО сразу предназначались для издательско-редакторской деятельности. Опередив всех своих конкурентов на целый уровень компьютерного развития и предоставив пользователю фантастические изобразительные возможности, фирма *Macintosh* допускает коммерческую недальновидность, ограничив доступ к разработкам приложений в своей операционной системе из вне. Этим, отвернув от себя основной состав программирующих специалистов всего мира.

Вскоре идею многооконной операционной системы подхватывает фирма *Microsoft*, выпустив графическую надстройку над операционной системой DOS – *Windows 3.1*. С неё начинается новый период в компьютерной графике.

На современных персональных компьютерах полностью используется графический режим, позволяющий моделировать различные шрифты для набора текста и математических формул, неограниченны возможности работы

с различными изображениями, и современному пользователю трудно представить себя без услуг, предоставляемых компьютерной графикой.

1.2. Этапы развития компьютерной графики

Итак, стартовав в 1950 г., компьютерная графика к настоящему времени прошла путь от экзотических экспериментов до одного из важнейших, всепроникающих инструментов современной цивилизации, начиная от научных исследований, автоматизации проектирования и промышленного производства, бизнеса, медицины, экологии, средств массовой информации, досуга и кончая бытовым оборудованием.

Можно выделить следующие этапы развития:

- *60-70-е годы научная дисциплина*. Бурное развитие методов, алгоритмов отсечение, генерация примитивных графических элементов, закраска узорами, реалистическое представление сцен (удаление невидимых линий и граней, трассировка лучей, излучающие поверхности);
- *80-е годы прикладная наука*. Отработка методов, средств, аппаратуры в различных сферах приложений;
- *90-е годы основное средство общения человека с ЭВМ*.

Направления компьютерной графики

Принято разделять компьютерную графику на следующие направления:

- *изобразительная компьютерная графика*,
- *обработка и анализ изображений*,
- *анализ сцен* (перцептивная компьютерная графика),
- *компьютерная графика для научных абстракций* (когнитивная компьютерная графика, способствующая познанию).

Изобразительная компьютерная графика

Объекты: синтезированные изображения.

Задачи:

- построение модели объекта и генерация изображения,
- преобразование модели и изображения,

- идентификация объекта и получение требуемой информации.

Обработка и анализ изображений

Объекты: дискретное, числовое представление фотографий.

Задачи:

- повышение качества изображения,
- оценка изображения определение формы, местоположения, размеров и других параметров требуемых объектов,
- распознавание образов выделение и классификация свойств объектов (обработка аэрокосмических снимков, ввод чертежей, системы навигации, обнаружения и наведения).

Анализ сцен

Предмет: исследование абстрактных моделей графических объектов и взаимосвязей между ними. Объекты могут быть как синтезированными, так и выделенными на фотоснимках. Первый шаг в анализе сцены выделение характерных особенностей, формирующих графический объект(ы).

Примеры: машинное зрение (роботы), анализ рентгеновских снимков с выделением и отслеживанием интересующего объекта, например, сердца. Итак, в основе анализа сцен (перцептивной компьютерной графики) находятся изобразительная графика + анализ изображений + специализированные средства.

Когнитивная компьютерная графика

Это компьютерная графика для научных абстракций, способствующая рождению нового научного знания. База мощные ЭВМ и высокопроизводительные средства визуализации.

Общая последовательность познания заключается в, возможно циклическом, продвижении от гипотезы к модели (объекта, явления) и решению, результатом которого является знание.

Первоначально ЭВМ имели малую производительность процессоров и средств компьютерной графики, т.е. по сути дела имели возможность работы

только с символами (некоторый упрощенный аналог логического мышления) С появлением *суперэвм*, производительностью в миллиард и более операций в секунду и графических *суперстанций*, производительностью до сотен миллионов операций в секунду, появилась возможность достаточно эффективного манипулирования образами (картинами). Важно отметить, что мозг не только умеет работать с двумя способами представления информации, причем с образами он работает иначе и эффективнее чем ЭВМ, но и умеет соотносить эти два способа и совершать (каким-то образом) переходы от одного представления к другому.

В этом контексте основная проблема и задача когнитивной компьютерной графики создание таких моделей представления знаний, в которых можно было бы однообразно представлять, как объекты, характерные для логического (символического, алгебраического) мышления, так и объекты, характерные для образного мышления.

Приложения компьютерной графики

Как уже отмечалось, компьютерная графика стала основным средством взаимодействия человека с ЭВМ. Важнейшими сформировавшимися областями приложений являются:

- компьютерное моделирование, которое явилось исторически первым широким приложением компьютерной графики,
- системы автоматизации научных исследований, системы автоматизации проектирования, системы автоматизации конструирования, системы автоматизации производства, автоматизированные системы управления технологическими процессами,
- бизнес,
- искусство,
- средства массовой информации,
- досуг.

В настоящее время появилось новое, очень интересное приложение компьютерной графики - виртуальная реальность.

По телевидению часто можно видеть передачи иллюстрирующие приложения компьютерной графики в автоматизации проектирования (были передачи об автоматизированном проектировании самолетов, автомобилей), много передач об автоматизации производства с различными робототехническими системами. Передачи о мире бизнеса практически не обходятся без показа различной дисплейной техники и ее использования. Что касается искусства, то достаточно упомянуть, что один из самых крупных первых суперкомпьютерных центров мира находился на студии Уолта Диснея и использовался для подготовки мультфильмов. Применение компьютерной графики в средствах массовой информации мы видим ежедневно, как в виде различных заставок и телеэффектов на экране, так и в виде газет, при подготовке многих из которых используется электронная верстка на компьютере.

2. ЦВЕТ

Цвет в компьютерной графике разбивается на два природных состояния: излучаемый источником света и отражаемый от поверхности. При этом, оба состояния базируются на физическом свойстве светового луча разлагаться на цветовой спектр, отличающийся длиной волны для каждого цвета. Человечеству удалось выделить основные цвета из многоцветного спектра, сочетание которых позволяет получать любые цветовые оттенки. Эти основные цвета являются основой для любой из существующих цветовых моделей и представлены для излучаемых или *аддитивных* моделей (RGB – red, green, blue) и отражаемых или *субтрактивных* моделей (CMY – cyan, magenta, yellow).

Принцип соответствия аддитивности цветного луча с субтрактивностью отражённого луча показан на рис.2.1.

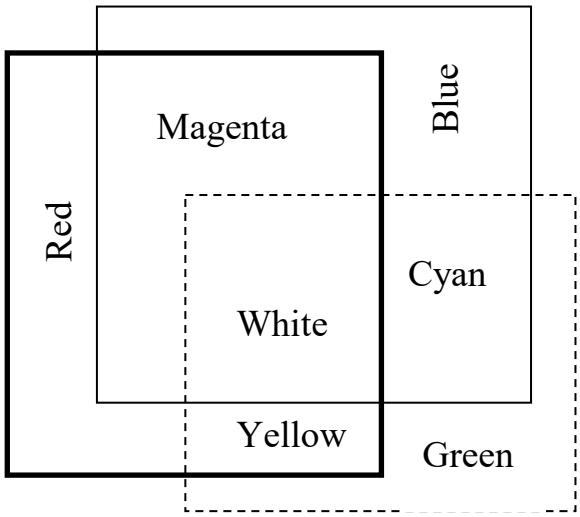


Рис.2.1.

Аддитивный цвет получается при соединении разных излучаемых цветов. В этой схеме отсутствие всех излучаемых цветов представляет собой чёрный (тьма), а присутствие всех излучаемых цветов - белый. Схема *аддитивных* цветов работает с излучаемым светом, например, монитор компьютера.

В схеме *субтрактивных* цветов происходит обратный процесс. Здесь получается какой-либо отражаемый цвет при вычитании других цветов из

общего луча света. В этой схеме белый цвет появляется в результате отсутствия всех отражаемых цветов, тогда как их присутствие даёт чёрный цвет. Схема *субтрактивных* цветов работает с отражённым светом.

В компьютерной графике применяют понятие цветового разрешения (другое название – глубина цвета). Оно определяет метод кодирования цветовой информации для ее воспроизведения на экране монитора. Для отображения черно-белого изображения достаточно двух бит (белый и черный цвета). Восьмиразрядное кодирование позволяет отобразить 256 градаций цветового тона. Два байта (16 бит) определяют 65 536 оттенков (такой режим называют High Color). При 24-разрядном способе кодирования возможно определить более 16,5 миллионов цветов (режим называют (True Color). С практической точки зрения цветовому разрешению монитора близко понятие цветового охвата. Под ним подразумевается диапазон цветов, который можно воспроизвести с помощью того или иного устройства вывода (монитор, принтер, печатная машина и прочие). В соответствии с принципами формирования изображения *аддитивным* или *субтрактивным* методами разработаны способы разделения цветового оттенка на составляющие компоненты, называемые цветовыми моделями. В компьютерной графике в основном применяют модели RGB и HSB для создания и обработки аддитивных изображений, а CMYK для печати изображения на полиграфическом оборудовании. Цветовые модели можно представить трехмерной системой координат, образующей цветовое пространство (рис. 2.2.). Из законов Германа Грассмана следует, что цвет можно выразить точкой в трехмерном пространстве. *Первый закон Грассмана* (*закон трехмерности*). Любой цвет однозначно выражается тремя составляющими, если они линейно независимы. Линейная независимость заключается в

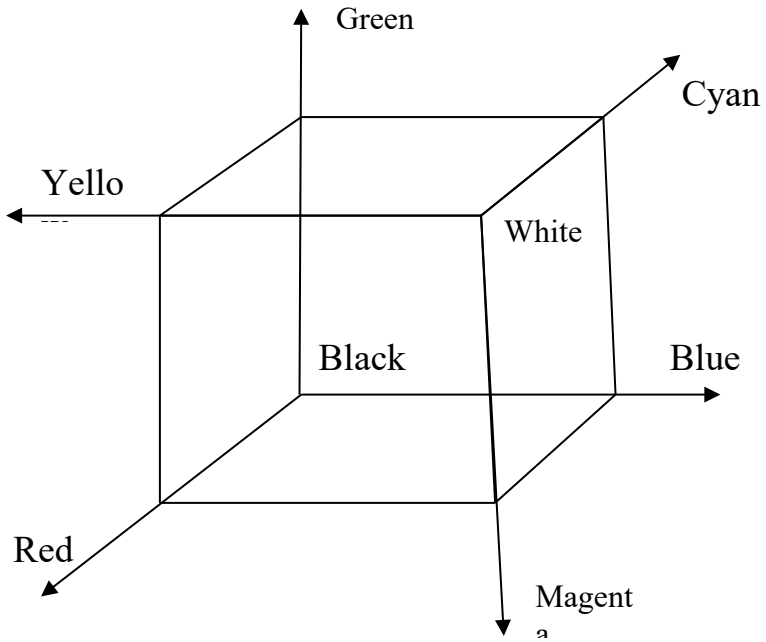


Рис.2.2.

невозможности получить любой из этих трех цветов сложением двух остальных.

Второй закон Германа Грассмана (*закон непрерывности*). При непрерывном изменении излучения цвет смеси также меняется непрерывно. *Не существует такого цвета, к которому нельзя было бы подобрать бесконечно близкий*.

Третий закон Грассмана (закон аддитивности). Цвет смеси излучений зависит только от их цвета, но не спектрального состава. То есть цвет (С) смеси выражается суммой цветовых уравнений излучений:

$$C_1 = r_1R + g_1G + b_1B$$

$$C_2 = r_2R + g_2G + b_2B$$

...

$$C_n = r_nR + g_nG + b_nB$$

$$C_{\text{сумм}} = (r_1 + r_2 + \cdots + r_n)R + (g_1 + g_2 + \cdots + g_n)G + (b_1 + b_2 + \cdots + b_n)B$$

где $r_i, g_i, b_i -$ коэффициенты интенсивности цвета, R, G, B – количество градаций палитры для соответствующего цвета.

Цветовая модель RGB

Монитор компьютера создает цвет непосредственно излучением света и, использует схему цветов RGB.

Цветовая модель RGB является аддитивной, то есть любой цвет представляет собой сочетание в различной пропорции трех основных цветов – красного (Red), зеленого (Green), синего (Blue). Она служит основой при создании и обработке компьютерной графики, предназначенной для электронного воспроизведения (на мониторе, телевизоре). Если с близкого расстояния посмотреть на экран монитора, то можно заметить, что он состоит из мельчайших точек красного, зелёного и синего цветов. Компьютер может управлять количеством света, излучаемого через любую окрашенную точку и, комбинируя различные сочетания любых цветов, может создать любой цвет. При наложении одного компонента основного цвета на другой яркость суммарного излучения увеличивается. Совмещение трех компонентов дает ахроматический серый цвет, который при увеличении яркости приближается к белому цвету. При 256 градационных уровнях тона черному цвету соответствуют нулевые значения RGB, а белому – максимальные, с координатами (255,255,255).

Цветовая модель CMYK

Модель CMYK имеет принцип наслоения цвета в процессе полиграфической печати. Сочетание всех трёх цветов при полной интенсивности образуют некоторый тёмный цвет, приближенный к чёрному. Те, кто хоть немного увлекается живописью, знают о том, что чистого чёрного цвета в природе не существует и ни один уважающий себя живописец не применяет этой краски для изображения пейзажа. В полиграфии этот цвет в первую очередь необходим для текстовой печати и введён как дополнительный четвёртый компонент модели. Но поскольку латинский символ «*B*» уже задействован под голубой цвет, то принято в аббревиатуру цветовой модели поставить последнюю букву слова «Blac**k**», а значит «*K*» - ***CMYK***.

3. ОСНОВЫ ПРОГАММИРОВАНИЯ ГРАФИЧЕСКИХ ПРИЛОЖЕНИЙ НА ЯЗЫКЕ C++

Первая историческая ступень программирования компьютерной графики базировалась на применении двухрежимного состояния вывода информации. Текстовую информацию обслуживали текстовые терминалы растрового принципа сканирования, которые представляли собой экран, поделённый на строки с делением на отдельные матрицы символов. Понятно, что при этом такие терминалы не отличались обилием текстовых атрибутов. Графическую информацию обслуживали векторные графические дисплеи, представлявшие исключительно векторный (отрезковый) тип изображения. С развитием телевизионных технологий на первый план выходят цветные растровые дисплеи, позволяющие работать сразу в двух режимах и обеспечившие появление новых графических моделей - растровых.

Код программы рисования диагонального отрезка через весь экран на языке С в системе MS-DOS выглядел таким образом:

```
main();
{   int X1, Y1, X2, Y2;
    initgr();
    X1 = 0; Y1 = 0;
    X2 = 640; Y2 = 480;
    move(X1, Y1);
    draw(X2, Y2);
    endgr();
}
```

При этом, программа содержит обращение к четырём графическим подпрограммам:

initgr() – инициирует графический вывод (переход дисплея в графический режим);

move(X1, Y1) – перемещает перо (реальное или фиктивное) в точку с координатами (X1, Y1);

draw(X2, Y2) – вычерчивает отрезок прямой линии из текущей позиции пера в точку с координатами (X2, Y2);

endgr() – выполняет заключительную операцию по выводу оставшейся информации из выходного буфера (осуществляется возврат к текстовому режиму дисплея).

С появлением современных компьютерных технологий компьютерная графика, как средство изображения человеческих ассоциаций легла в основу организации диалога между человеком и компьютером. Базируясь на изобразительных преимуществах, операционная система полностью отдаёт предпочтение графическому режиму, приобретая для текстового вывода широту стилевых возможностей и, самое главное, реализация многозадачности через многооконный режим.

Стиль программирования Windows-приложений принципиально отличается от программирования в системах раннего поколения. В MS-DOS программа монопольно владеет всеми ресурсами системы и является инициатором взаимодействия с операционной системой. Операционная система Windows, задуманная как многозадачная, является инициатором обращения к программе. Все ресурсы Windows являются разделяемыми, и Windows-приложение не может владеть ими монопольно. Windows-приложение ожидает посылки сообщения операционной системой, и лишь после его получения, выполняет определённое действие, а затем вновь переходит в режим ожидания очередного сообщения от операционной системы.

На рисунке 3.1. схематично изображена диаграмма типичного *Windows-приложения.*

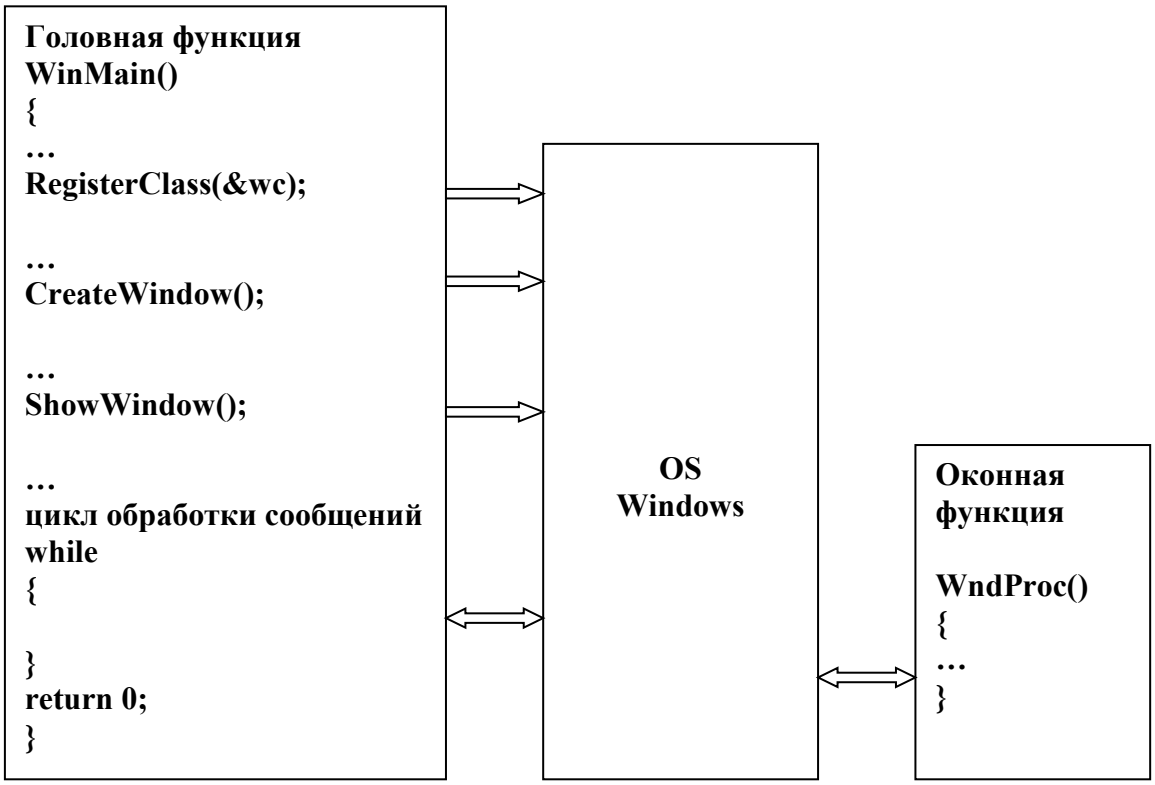


Рис.3.1. Структура Windows-приложения

В отличие от программы, выполненной в операционной системе *MS-DOS*, для создания простейшего приложения под *Windows* надо проделать намного больше работы. Чтобы работать с оконным интерфейсом, *Windows-приложение* должно выполнять основные стандартные действия:

1. Определить класс окна.

2. Зарегистрировать окно.

3. Создать окно данного класса.

4. Отобразить окно.

5. Запустить цикл обработки сообщений.

Создадим минимальное *Win32*-приложение загрузив Visual Studio 2010. Выполним команду *File* | *New* | *Project…* и выберем тип проекта − *Win32 Project*. В раскрывающемся списке *Location* выберем путь к рабочей папке, а в поле *Name* имя проекта (рис.3.2). В следующем диалоговом окне, приведённом на рисунке 3.3, нажимаем кнопку *Next*, а в окне опций проекта (3.4) выберем

флажок *Empty project* (Пустой проект) и нажав кнопку *Finish* – получаем пустой проект, в котором нет ни одного файла.

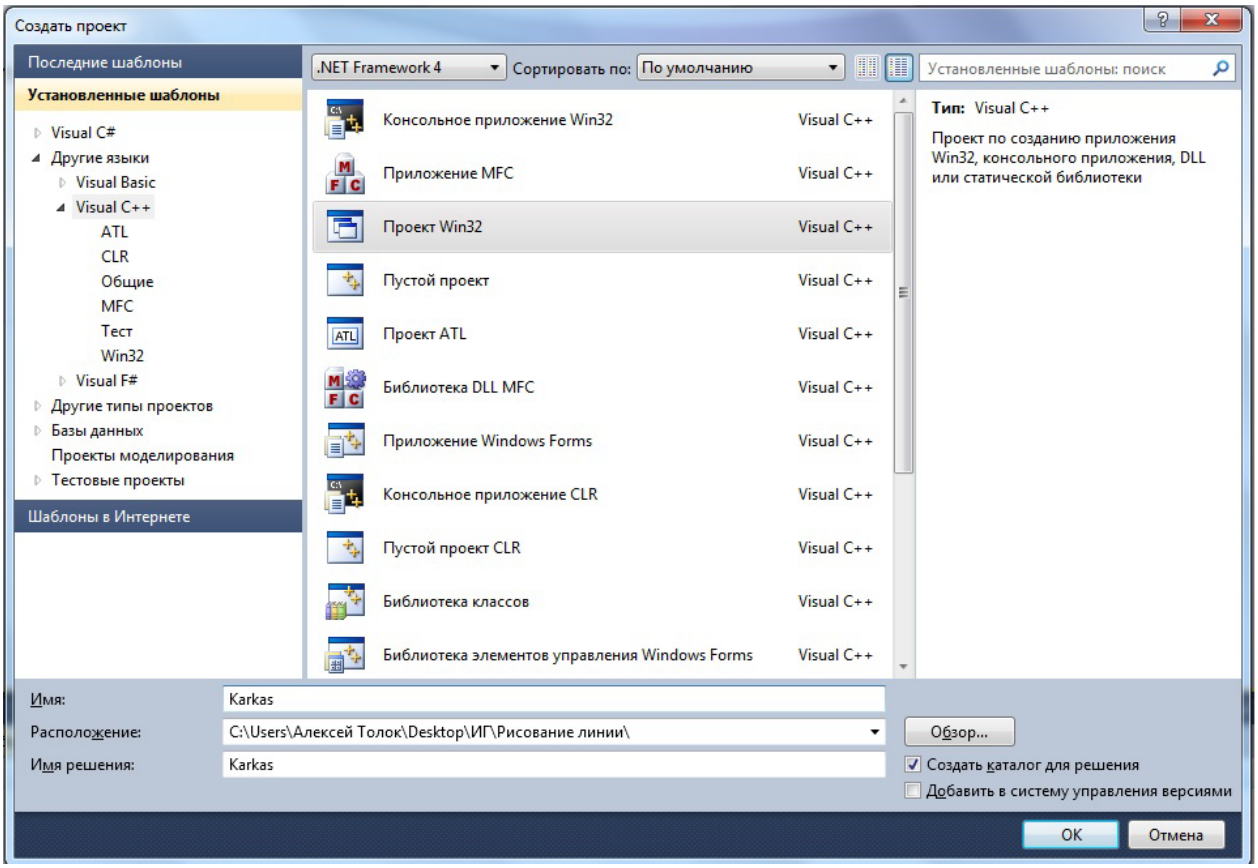


Рис.3.2. Выбор типа проекта.

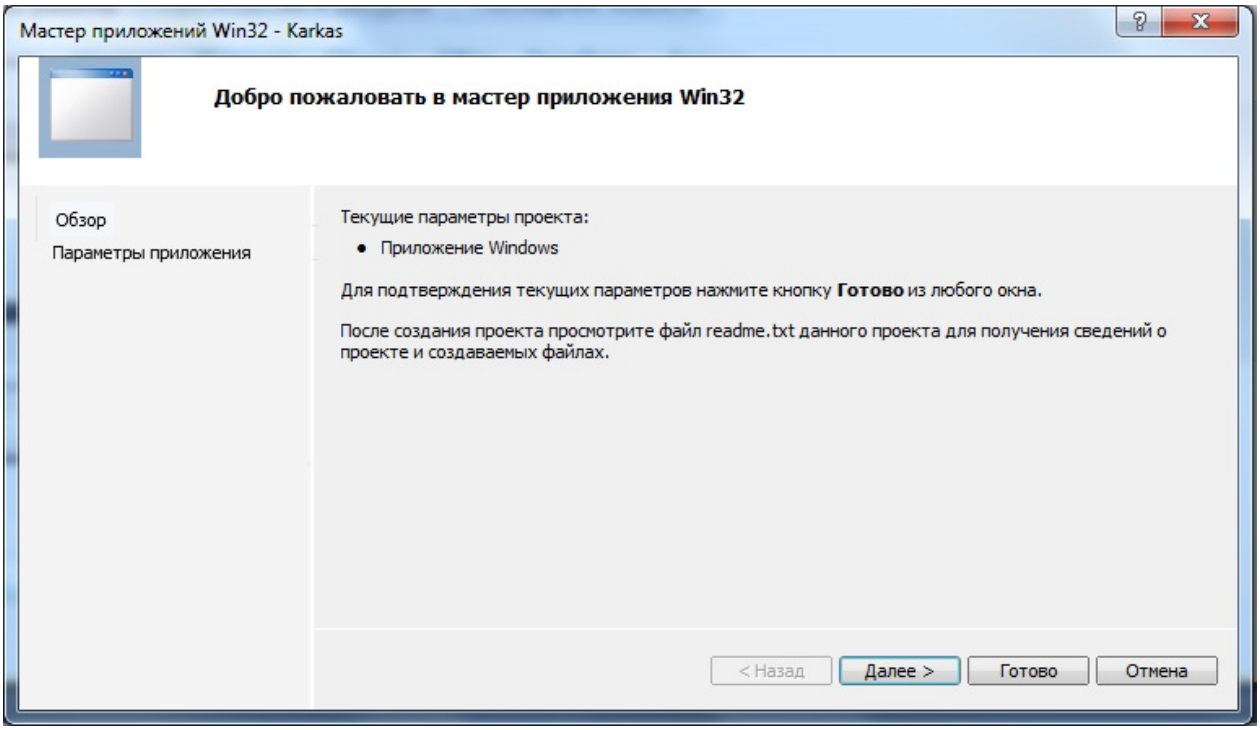


Рис.3.3. Стартовое окно построителя приложения.

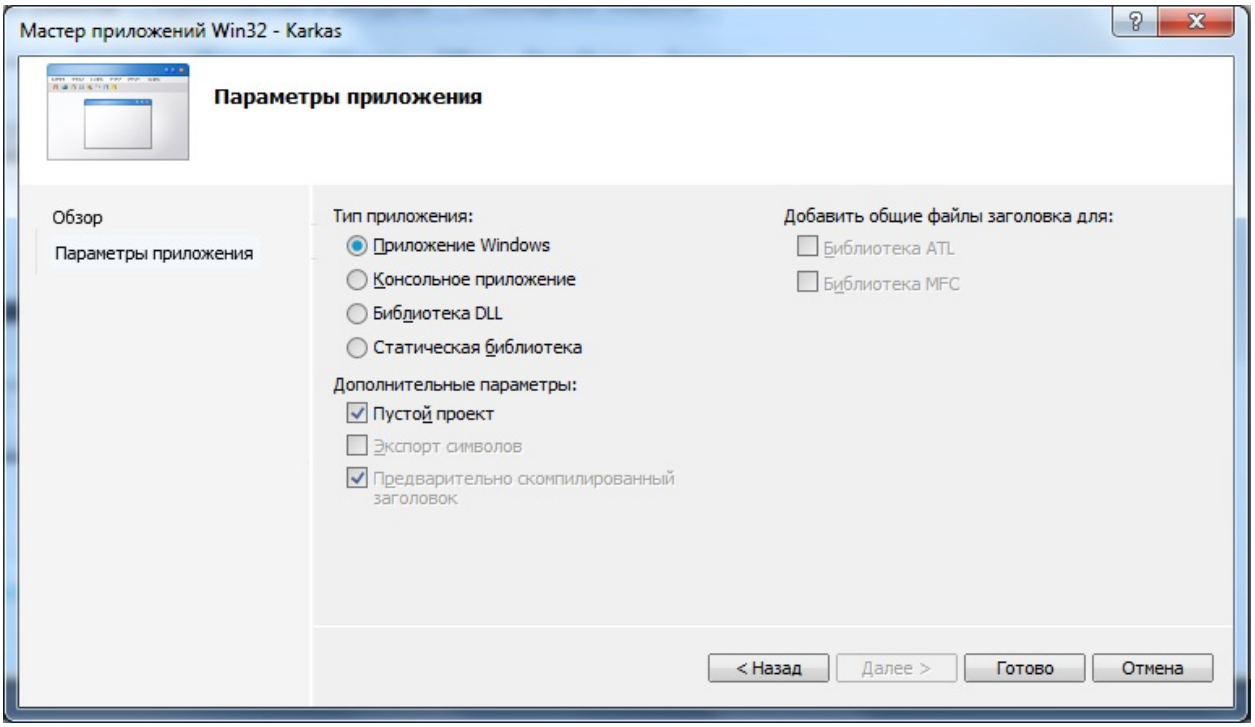


Рис.3.4. Окно опций проекта

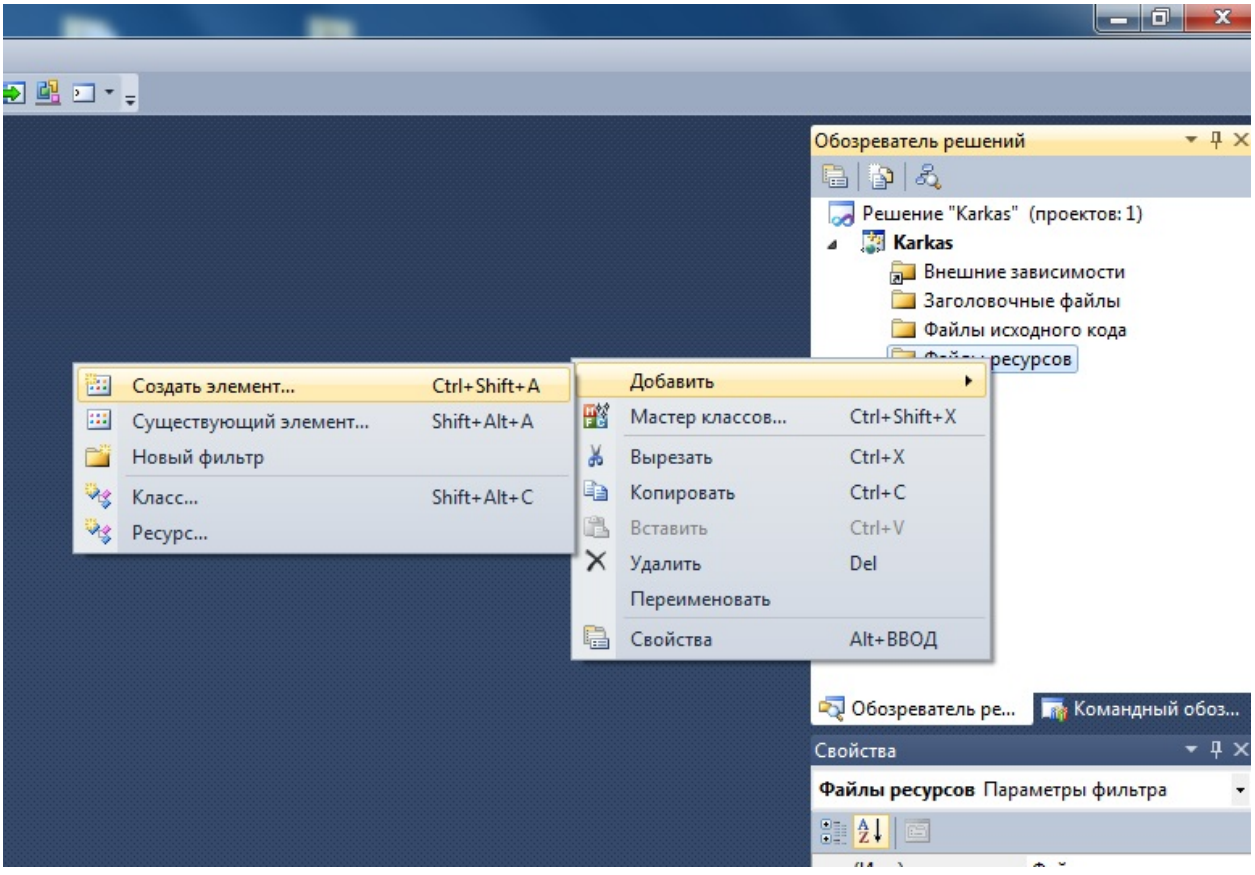


Рис.3.5. Добавление к проекту нового объекта с помощью контекстного меню

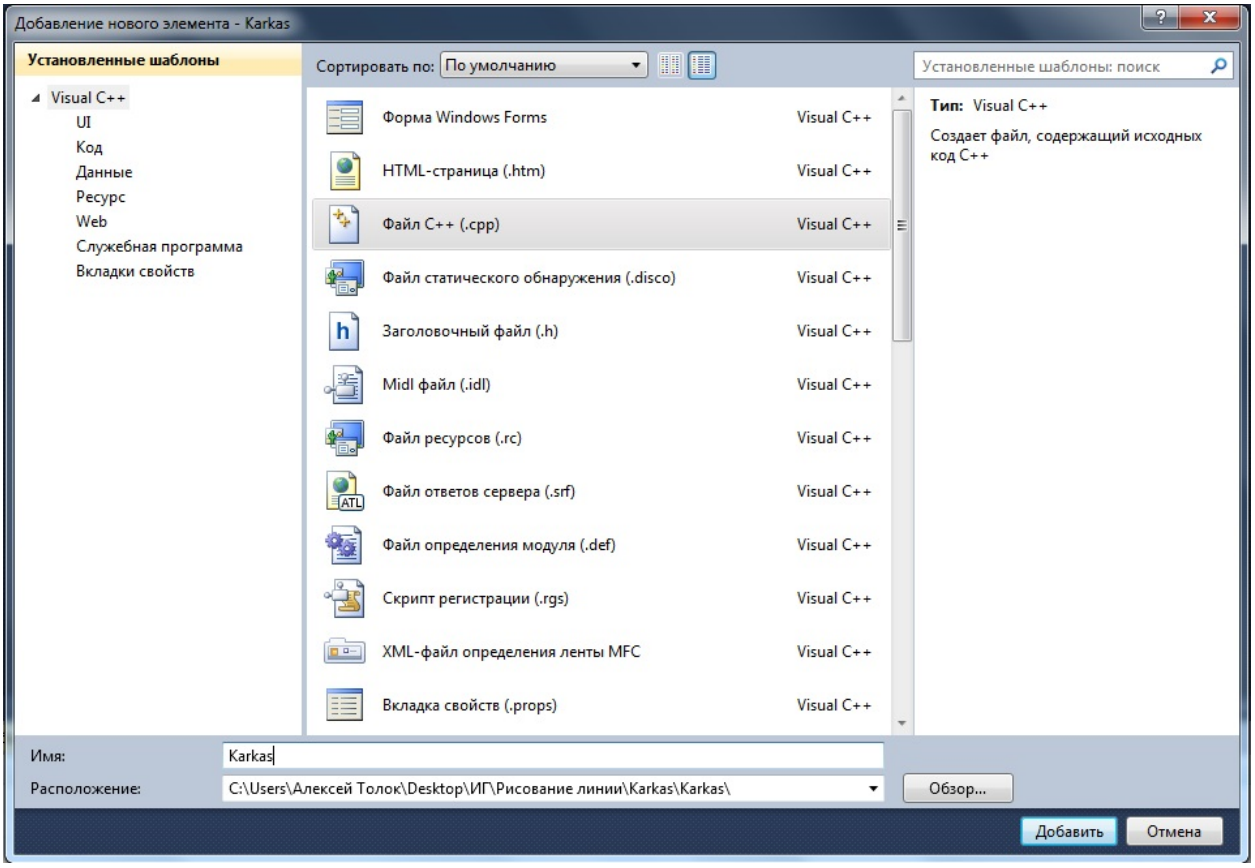


Рис.3.6. Выбор шаблона объекта

С помощью контекстного меню (рис.3.5) добавим файл C++ для ввода кода приложения, имя файла «Karkas» введём в ходе диалога выбора шаблона объекта согласно рис.3.6.

С помощью приведённого ниже листинга 3.1 рассмотрим основной «каркас» *Windows*-приложения.

Листинг 3.1. Минимальный код каркаса Windows-приложения

```
//---------------------------------------------------------------------------
#include <windows.h>
#include <tchar.h>
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
TCHAR WinName[] = _T("MainFrame");
//---------------------------------------------------------------------------
int APIENTRY WinMain(HINSTANCE This, HINSTANCE Prev, LPSTR cmd,
        int mode)
```

```
{
HWND hWnd;        //Дескриптор главного окна программы
MSG msg;          // Структура для хранения сообщений
WNDCLASS wc;      // Класс окна
// Определение класса окна
wc.hInstance =  This;
wc.lpszClassName = WinName;   // Имя класса окна
wc.lpfnWndProc = WndProc;     // Функция окна
wc.style = CS_HREDRAW | CS_VREDRAW; // Стиль окна
wc.hIcon = LoadIcon(NULL, IDI_APPLICATION); //Стандартная иконка
wc.hCursor = LoadCursor(NULL, IDC_ARROW); //Стандартный курсор
wc.lpszMenuName = NULL;         //Нет меню
wc.cbClsExtra = 0;              //Нет дополнительных данных класса
wc.cbWndExtra = 0;              //Нет дополнительных данных окна
// Заполнение окна белым цветом
wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
if(!RegisterClass(&wc)) return 0; // Регистрация класса окна
// Создание окна
hWnd = CreateWindow(WinName, //Имя класса окна
_T("Каркас Windows-приложения"),   // Заголовок окна
WS_OVERLAPPEDWINDOW,               // Стиль окна
CW_USEDEFAULT,     // X
CW_USEDEFAULT,     // Y
CW_USEDEFAULT,     // Width
CW_USEDEFAULT,      // Height
HWND_DESKTOP,       // Дескриптор родительского окна
NULL,               // Нет меню
This,               // Дескриптор приложения
NULL);              // Дополнительной информации нет
ShowWindow(hWnd, mode); // Показать окно
```

```
 // Цикл обработки сообщений
 while(GetMessage(&msg, NULL, 0, 0))
 {
 TranslateMessage(&msg); // Функция трансляции кодов нажатой клавиши
 DispatchMessage(&msg);  // Посылает сообщение функции WndProc()
 }
        return 0;
}
//--------------------------------------------------------------------
// Оконная функция вызываемая операционной системой
// и получает сообщения из очереди для данног приложения
LRESULT CALLBACK WndProc(HWND hWnd, UINT message,
                                        WPARAM wParam, LPARAM lParam)
{
//Обработчик сообщений
switch(message)
{
case WM_DESTROY : PostQuitMessage(0);
                  break; // Завершение программы
                  // Обработка сообщения по умолчанию
default : return DefWindowProc(hWnd, message, wParam, lParam);
}
 return 0;
}
```

Программа не делает ничего полезного, поэтому, запустив её на выполнение кнопкой (*Start Debagging*), получим изображённое на рисунке 3.7 пустое окно, имеющее заголовок и набор стандартных кнопок.

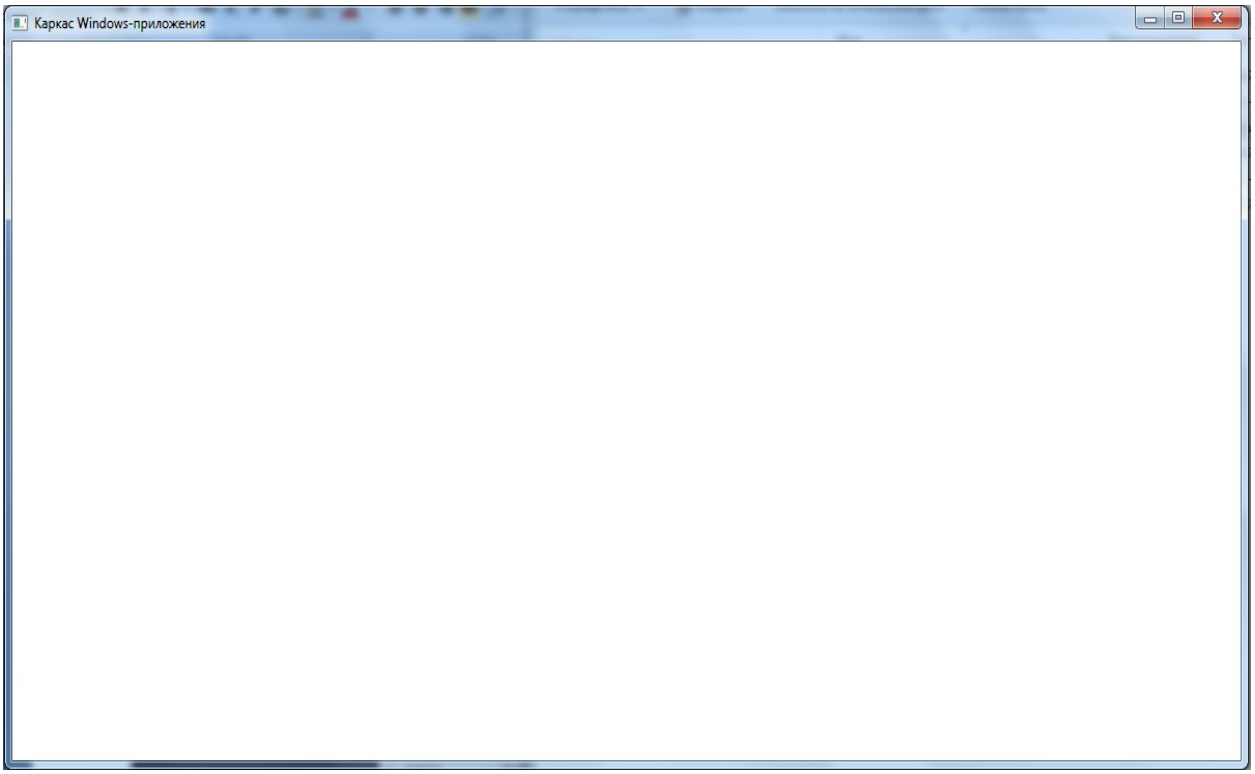


Рис.3.7. Окно первой Windows-программы

Далее умышленно опускаются тонкости приведённого программного кода. Рассмотрим блочно архитектуру программы, выделив лишь самое необходимое для возможности построения интерактивных изображений в полученном окне приложения.

Расставленные комментарии позволяют выделить смысловые блоки кода, представленного листингом 3.1. В начале кода присутствует описательная часть программы в которую входит подключение двух библиотек *windows.h* и *tchar.h*. Первая из упомянутых библиотек является основной и присутствует во всех *Windows*-приложениях. Он служит для вызова других файлов включений, основные из которых: *windef.h, winbase.h, wingdi.h, winuser.h*; а также несколько дополнительных файлов, в которых помещены определения *API*-функций, констант и макросов. В файле *tchar.h* содержатся определения некоторых полезных макросов для работы со строковыми типами данных.

Далее следует прототип оконной функции:

```
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
```

Оконная функция также является функцией обратного вызова, что связано с некоторыми особенностями организации вызовов операционной системы. Эта функция регистрируется в системе, а её вызов осуществляет операционная система, когда требуется обработать сообщение. Тип возвращаемого значения функции LRESULT эквивалентен *long* для *Win*32-проекта.

На глобальном уровне описывается имя класса окна приложения в виде текстовой строки:

TCHAR WinName[] = _T("MainFrame");

Имя класса окна используется операционной системой для его идентификации. Имя может быть произвольным, в частности содержать кириллический текст.

Внутри головной функции *WinMain()* описаны три переменных:

hWnd – предназначена для хранения дескриптора главного окна программ;

msg – это структура, в которой хранится информация о сообщении, передаваемом операционной системой окну приложения:

```
struct MSG
{
HWND hWnd;              // Дескриптор окна
UINT message;           // Номер сообщения
WPARAM wParam;          // 32-разрядные целые содержат
LPARAM lParam;          // дополнительные параметры сообщения
DWORD time;             // Время посылки сообщения в миллисекундах
POINT pt;               // Координаты курсора (x, y)
   };
   struct POINT
  {
        LONG x, y;   };
```

wc – структура, содержащая информацию по настройке окна.

Определив значения полей оконной структуры *wc* создаётся окно при помощи *API*-функции *CreateWindow(),* где стоит обратить внимание на параметр *_T(“Каркас Windows-приложения”).* В двойных кавычках задаётся смысловое название приложения, выводимое в верхней левой части рамки окна. Окно создано, но пока не отображается. Для отображения окна применяется функция *ShowWindow().*

Заключительная часть головной функции – цикл обработки сообщений. он задаётся оператором *wile*, аргументом которого является функция *GetMessage()*. Такой цикл является обязательным для всех *Windows*-приложений, его цель – получение и обработка сообщений, передаваемых операционной системой. Операционная система ставит сообщения в очередь, откуда они извлекаются функцией *GetMessage()* по мере готовности приложения. Сообщения определяются их номерами, символические имена для них определены в файле включений *winuser.h*. тпрефикс всех системных сообщений *WM_*.

Внутри цикла расположены две функции:

TranslateMessage(&msg);

DispatchMessage(&msg);

Первая из них транслирует код клавиши в клавиатурные сообщения *WM_CHAR*.

Вторая функция обеспечивает возврат преобразованного сообщения обратно операционной системе и инициирует вызов оконной функции данного приложения для его обработки.

Основным компонентом функции *WndProc()* является переключатель *switch*, обеспечивающий выбор соответствующего обработчика сообщений по его номеру *message*. В рассматриваемом случае предусмотрена обработка лишь одного сообщения *WM_DESTROY*. Это сообщение посылается когда пользователь завершает программу. Получив его, огконная функция вызывает функцию *PostQuitMessage(0)*, которая завершает приложение и передаёт операционной системе код возврата – 0. Говоря точнее, генерируется

сообщение *WM_QUIT*, получив которое функция *GetMessage()* возвращает нулевое значение. В результате цикл обработки сообщений прекращается и происходит завершение работы приложения.

Отрезок.

Если условиться, что стандартное оформление окна будущего приложения нас устраивает и основное внимание уделить содержимому этого окна, то оставим основную часть кода листинга 3.1 без изменений, сконцентрировавшись, лишь на обработчике сообщений функции *WndProc()*.

На листинге 3.2 показан фрагмент кода, позволяющий изобразить диагональный отрезок через всё окно приложения аналогично примеру, представленному программой на языке С в системе MS-DOS в начале раздела.

Листинг 3.2. Фрагмент кода на C++ для отрисовки диагонального отрезка через всё окно приложения:

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message,
                         WPARAM wParam, LPARAM lParam)
{
PAINTSTRUCT ps;
        HDC hdc;
        static int sx, sy;
        static HPEN hpen;
//Обработчик сообщений
switch(message)
{
case WM_CREATE:
                hpen = CreatePen(PS_SOLID, 2, RGB(0, 0, 255));
                break;
case WM_SIZE:
                sx = LOWORD(lParam); //Ширина окна в экранных координатах
                sy = HIWORD(lParam); //Высота окна в экранных координатах
             break;
```

```
case WM_PAINT:
        hdc = BeginPaint(hWnd, &ps); //Начало рисования
        SelectObject(hdc, hpen); //Синее перо
        //Изображение диагонального отрезка
        MoveToEx(hdc, 0, 0, NULL); //начало отрезка
        LineTo(hdc, sx, sy); //завершение отрезка
        EndPaint(hWnd, &ps);   //завершение рисования
        break;
case WM_DESTROY :
        DeleteObject(hpen);
        PostQuitMessage(0);
        break; // Завершение программы
        // Обработка сообщения по умолчанию
default : return DefWindowProc(hWnd, message, wParam, lParam);
}
 return 0;
}
```

Рассмотрим вывод изображения отрезка в окно приложения в сообщении *WM_PAINT*. Для этого необходимо получить *контекст устройства.* В *Windows* все функции, выводящие что-либо в окно, используют в качестве параметра *дескриптор контекста устройства hdc*, который представляет собой структуру, описывающую свойства данного устройства вывода. В оконной функции опишем эту переменную:

HDC hdc;

В обработчике *WM_PAINT* вызовом функции *BeginPaint()* получим *hdc.*

Вся необходимая информациядля перерисовки окна будет представлена в структуре *PAINTSTRUCT*:

```
struct PAINTSTRUCT {
HDC      hdc;              // Контекст устройства;
```

BOOL fErase; // Если TRUE – фон окна перерисовывается;

RECT rcPaint; // Недействительный прямоугольник;

BOOL fRestore; // Резерв;

BOOL fIncUpdate; // Резерв;

BYTE rgbReserved[32]; // Резерв:

};

Теперь можно выводить отрезок в окно с помощью функций *MoveToEx()* и *LineTo()*. Параметры этих функций *sx* и *sy* содержат размер окна приложения и определяются в обработчике *WM_SIZE*.

Рисование линии

Для того, чтобы нарисовать линию, используется функция *LineTo()*:

BOOL WINAPI LineTo(HDC hdc, int x, int y);

Здесь *hdc* – дескриптор контекста устройства, *x, y* – координаты конца линии. Начало линии определяется текущей позицией рисования. При создании окна текущая позиция определяется в начале координат.

Установка текущей позиции

Для установки точки в текущую позицию с координатами *x, y* применяется функция:

BOOL WINAPI MoveToEx(HDC hdc, int x, int y, LPPOINT oldPoint);

Если последний параметр *NULL*, то предыдущие координаты не сохраняются.

Определение размера клиентской области

Имеется достаточно простое решение для определения клиентской области окна. Сообщение *WM_SIZE* генерируется системой при создании окна после сообщения *WM_CREATE* и при каждом изменении его размеров:

case WM_SIZE:

sx = LOWORD(lParam); // ширина окна

sy = HIWORD(lParam); // высота окна

break;

Рисование прямоугольника

Нарисовать прямоугольник можно при помощи функции:

BOOL WINAPI Rectangle(HDC hdc, int x1, int y1, int x2, int y2);

где *(x1, y1)* – координаты левого верхнего угла прямоугольника, а *(x2, y2)* – координаты правого нижнего угла.

Рисование эллипса

Функция для изображения эллипса имеет те же параметры, поскольку эллипс определяется ограничивающим его прямоугольником

BOOL WINAPI Ellipse(HDC hdc, int x1, int y1, int x2, int y2);

Рисование точки

Функция, выводящая в окно одну точку описана как:

COLORREF WINAPI SetPixel(HDC hdc, int x, int y, COLORREF color);

Зададим функцию рисования красного пикселя:

SetPixel(hdc, x, y, RGB(255, 0, 0));

Создание пера

Пользовательское перо создаётся функцией:

HPEN WINAPI CreatePen(int style, int width, COLORREF color);

Параметр *style* задаёт тип линии, приведённой в таблице:

Макрос	**Тип линии**
PS_DASH	Штриховая линия
PS_DASHDOT	Штрихпунктирная линия
PS_DASHDOTDOT	Двух штрихпунктирная линия
PS_DOT	Точечная линия
PS_NULL	Прозрачное перо
PS_SOLID	Сплошная линия

Параметр *width* задаёт ширину пера в логических единицах. Если ширина пера установлена в ноль, то линии рисуются в один пиксель, независимо от режима рисования.

После создания перо выбирается как текущее с помощью функции:

HGDIOBJ WINAPI SelectObject(HDC hdc, HGDIOBJ handle);

Создать синее перо толщиной в 2 пикселя можно таким образом:

HPEN hpen1=CreatePen(PS_SOLID, 2, RGB(0, 0, 255));

SelectObject(hdc, hpen1);

Создание кисти

Кисть используется для заполнения фона окна или замкнутой области внутри окна.

Сплошная кисть создаётся при помощи функции:

HBRUSH Create SolidBrush(COLORREF color);

Устанавливается функцией *SelectObject()* , например:

HBRUSH hbrush = CreateSolidBrush(RGB(255, 255, 0));

SelectObject(hdc, hbrush);

Кисть может быть удалена функцией *DeleteObject()*.

Создание пути

Путь – это набор прямых линий и кривых. Его можно применять для графических построений сложных регионов и использовать для отсечения.

HDC WINAPI BeginPath(HDC hdc);

Открывает путь, теперь графические функции в окно ничего не выводят, а строят путь.

HDC WINAPI CloseFigure(HDC hdc);

Закрывает открытую фигуру в пути. Замыкает первую и последнюю точки.

HDC WINAPI EndPath(HDC hdc);

Закрывает путь и помещает его в контекст устройства.

HDC WINAPI FillPath(HDC hdc);

Закрашивает текущей кистью область, ограниченную путём.

HDC WINAPI PathToRegion(HDC hdc);

Преобразует путь в область, возвращает его дескриптор.

HDC WINAPI StrokePath(HDC hdc);

Обводит путь текущим пером.

На рисунке 3.8. показан пример работы пути при создании изображения звезды.

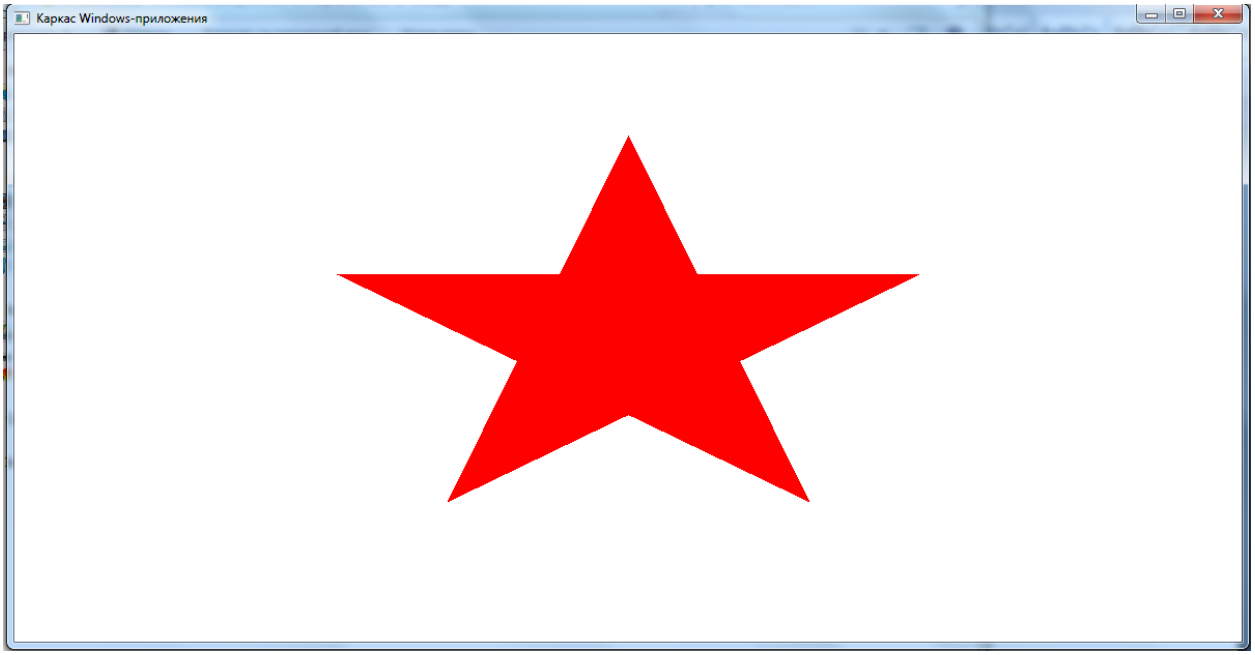


Рис.3.8. Звезда

4. ДВУХМЕРНЫЕ АЛГОРИТМЫ

4.1. Преобразование и новые координаты

Рассмотрим следующую систему уравнений:

$$\begin{cases} x' = x + a \\ y' = y \end{cases}$$

Эти уравнения можно интерпретировать двояким способом:

1. Все точки на плоскости x и y перемещаются влево на расстояние a - см. рис 4.1.1(а)

2. Координатные оси x и y перемещаются влево на расстояние a - см. рис. 1.1.1(б)

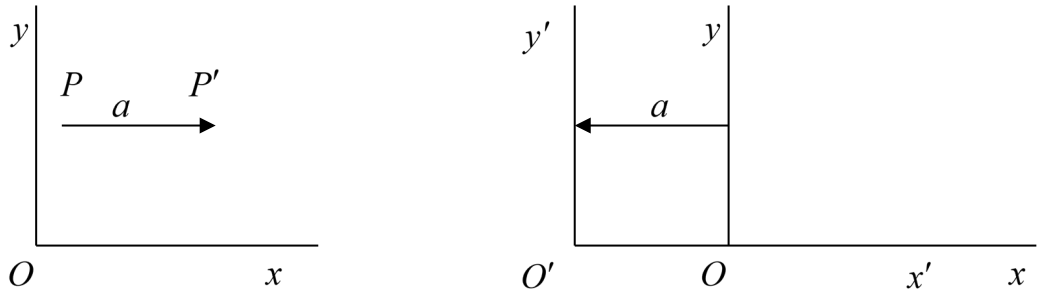


Рис.4.1.1. (а) – перенос; (б) – изменение координат

Этот простой пример иллюстрирует принцип, применимый и в более сложных ситуациях. Мы и далее будем рассматривать системы уравнений, обычно записываемых в виде произведения матриц, интерпретируя их как преобразования всех точек в фиксированной системе координат. Однако та же самая система уравнений может интерпретироваться как изменение системы координат.

Пусть необходимо повернуть точку $P(x,y)$ вокруг начала координат O на угол φ. Изображение новой точки на рис. 4.1.2 обозначим через $P'(x',y')$. Существует четыре числа a,b,c,d, такие, что новые координаты x' и y' могут быть вычислены по значениям старых координат x и y из следующих уравнений:

$$\begin{cases} x' = ax + bY \\ y' = cx + dy \end{cases} \qquad (4.1.1)$$

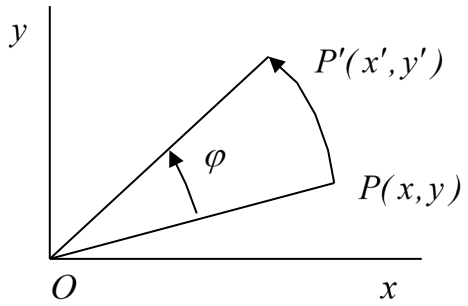


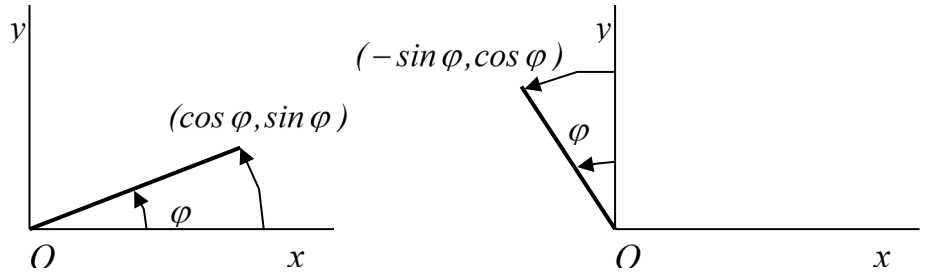
Рис.4.1.2. Поворот вокруг точки O на угол φ

Для получений значений a,b,c,d рассмотрим в начале точку $(x, y) = (1,0)$. Полагая $x = 1$ и $y = 0$ в уравнении (4.1.1), получим

$$x' = a$$
$$y' = c$$

Но в этом простом примере, как это видно из рис. 4.1.3(а), значения x' и y' равны соответственно $\cos\varphi$ и $\sin\varphi$. Тогда будем иметь

$$a = \cos\varphi$$
$$c = \sin\varphi$$



а) б)

Рис.4.1.3. (а) – отображение точки (1,0); (б) – отображение точки (0,1)

Аналогичным образом из рис. 1.3(б) следует

$$b = -\sin\varphi$$
$$d = \cos\varphi$$

Тогда вместо системы уравнений (4.1) можем записать

$$\begin{cases} x' = x\cos\varphi - y\sin\varphi \\ y' = x\sin\varphi + y\cos\varphi \end{cases} \qquad (4.1.2)$$

В приведённом ниже фрагменте программы изображение стрелки вычерчивается после предварительного расчёта координат поворотом на угол 6^o вокруг начала координат:

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message,
                                    WPARAM wParam, LPARAM lParam)
{
 PAINTSTRUCT ps;
 HDC hdc;
 static int sx, sy;
 static HPEN hpen;
 int a, b, x_scr, y_scr;    //Экранные координаты
      float x[4]={6.0, 6.0, 5.9, 6.1},
           y[4]={-0.25, 0.25, 0.0, 0.0}; //Объектные координаты стрелки
      float xMax=6.5, yMax=6.5;  //Обеъектные координаты окна
      float Kx, Ky;              //Коэффициенты масштабирования объектных
                                 //координат в экранные
//Обработчик сообщений
switch(message)
{
case WM_CREATE:
          hpen = CreatePen(PS_SOLID, 2, RGB(0, 0, 0));
          break;
case WM_SIZE:
          sx = LOWORD(lParam); //Ширина окна в экранных координатах
          sy = HIWORD(lParam); //Высота окна в экранных координатах
             break;
case WM_PAINT:
          hdc = BeginPaint(hWnd, &ps); //Начало рисования
             int i,j;
```

```
            float pi, phi, cos_phi, sin_phi, xx, yy;
            pi=4.0*atan(1.0); phi=6*pi/180;      //определение угла поворота
            cos_phi=cos(phi); sin_phi=sin(phi);  //в радианах
        SelectObject(hdc, hpen); //Синее перо
            Kx=sx/xMax;
            Ky=sy/yMax;
            for(i=1; i<=14; i++)  //Цикл отрисовки 14 стрелок
            {
                for(j=0; j<=3; j++)  //Цикл пересчёта координат поворота
                {                    //стрелки
                    xx=x[j]; yy=y[j];
                    x[j]=xx*cos_phi-yy*sin_phi;
                    y[j]=xx*sin_phi+yy*cos_phi;
                }
              //Изображение текущей стрелки
              MoveToEx(hdc, x[0]*Kx, sy-y[0]*Ky, NULL); //начало стрелки
              for(j=1; j<=3; j++) LineTo(hdc, x[j]*Kx, sy-y[j]*Ky);
              LineTo(hdc, x[1]*Kx, sy-y[1]*Ky); //завершение стрелки
            }
        EndPaint(hWnd, &ps);   //завершение рисования
        break;
case WM_DESTROY :
        DeleteObject(hpen);
        PostQuitMessage(0);
        break; // Завершение программы
        // Обработка сообщения по умолчанию
default : return DefWindowProc(hWnd, message, wParam, lParam);
}
 return 0;
}
```

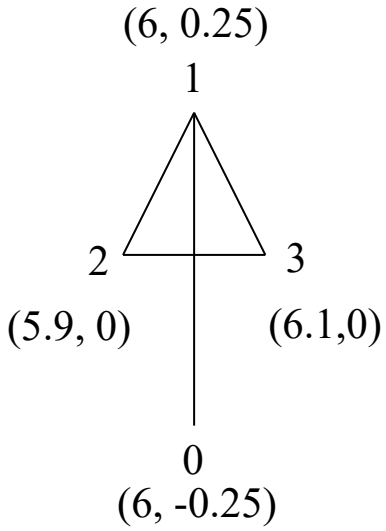


Рис.4.1.4. Описание координат стрелки

Начальная позиция стрелки изображена на рис. 4.1.4. Стрелка указывает вверх и её центр располагается в точке (6, 0). Значения координат вершин ломанной, изображающей стрелку, записаны в элементах массивов $x[i]$ и $y[i]$ $(i = 0,1,2,3)$. Индексация элементов массива начинается с нуля. В описании массива необходимо задать количество элементов массива. Следовательно, объявление массива типа $float\ x[4] =$ требует введения значений четырёх элементов:

$$float\ x[4] = \{6.0, 6.0, 5.9, 6.1\};$$

Тот факт, что начальное значение $y[0]$ задаётся отрицательным при положительных координатах рабочего поля объекта не должен смущать, поскольку первая же стрелка в цикле претерпевает поворот на 6°, перемещаясь в положительную область значений.

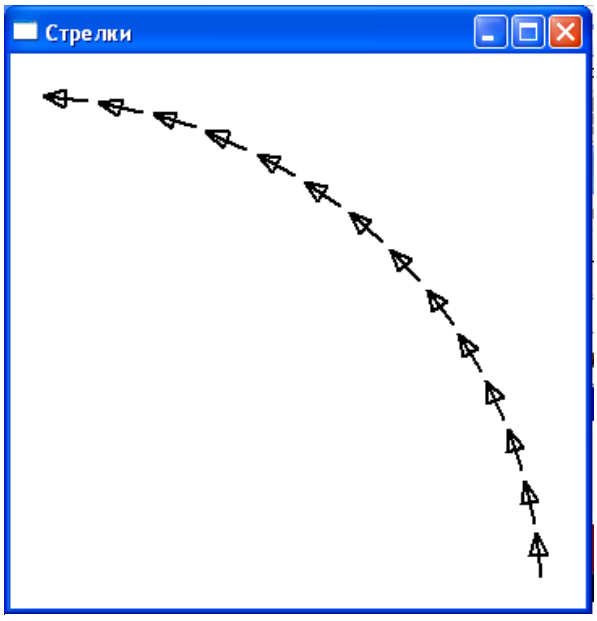


Рис.4.1.5. Результат компьютерного изображения поворота стрелки

Для возможности компоновки объекта соответственно рабочему полю выводимого окна применяются коэффициенты масштабирования Kx, Ky, определяемые через отношение габаритов рабочего поля объекта $xMax, yMax$ (поля, ограничивающего реально заданные размеры изображаемого объекта) и габаритов рабочего поля выводимого окна на экране дисплея Sx, Sy.

Результат работы программы представлен на рисунке 4.1.5.

4.2. Поворот

Система уравнений (4.1.2) описывает поворот вокруг точки O - начала системы координат. Но часто это не то, что нам нужно. Если требуется выполнить поворот относительно заданной точки (x_0, y_0), то в этих уравнениях можно заменить : x на $x - x_0$, y - на $y - y_0$, x' - на $x' - x_0$ и y' - на $y' - y_0$:

$$\begin{cases} x' - x_0 = (x - x_0)\cos\varphi - (y - y_0)\sin\varphi \\ y' - y = (x - x_0)\sin\varphi + (y - y_0)\cos\varphi \end{cases} \qquad (4.2.1)$$

$$\begin{cases} x' = x_0 + (x - x_0)\cos\varphi - (y - y_0)\sin\varphi \\ y' = y_0 + (x - x_0)\sin\varphi + (y - y_0)\cos\varphi \end{cases}$$

Приведённый ниже фрагмент программы демонстрирует способ описания вращения стрелки вокруг выбранной точки (x_0, y_0). Результат работы приложения представлен рисунком 4.2.1.

```
…
float x[4]={0.0, 0.0, -0.08, 0.08},
      y[4]={-0.25, 0.25, 0.0, 0.0}; //Объектные координаты стрелки
float xMax=8.5, yMax=8.5;  //Обеъектные координаты окна
float Kx, Ky;              //Коэффициенты масштабирования объектных
                           //координат в экранные
float x0=4.5, y0=4.5, r=3.0;//Координаты центра и радиус окружности
…
      //Перенос в начальную позицию (x0+r,y0)
for(j=0;j<4;j++){x[j]+=x0+r; y[j]+=y0;}
for(i=0;i<30;i++)
{
      for(j=0;j<4;j++)//Цикл пересчёта координат текущей стрелки
            { dx=x[j]-x0; dy=y[j]-y0;//Сдвиг по осям в точку (x0,y0)
              x[j]=x0+dx*cos_phi-dy*sin_phi;  y[j]=y0+dx*sin_phi+dy*cos_phi; }
```

```
    MoveToEx(hdc, x[0]*Kx, sy-y[0]*Ky, NULL); //начало стрелки
     for(j=1; j<=3; j++) LineTo(hdc, x[j]*Kx, sy-y[j]*Ky)
    LineTo(hdc, x[1]*Kx, sy-y[1]*Ky); //завершение стрелки
}
```

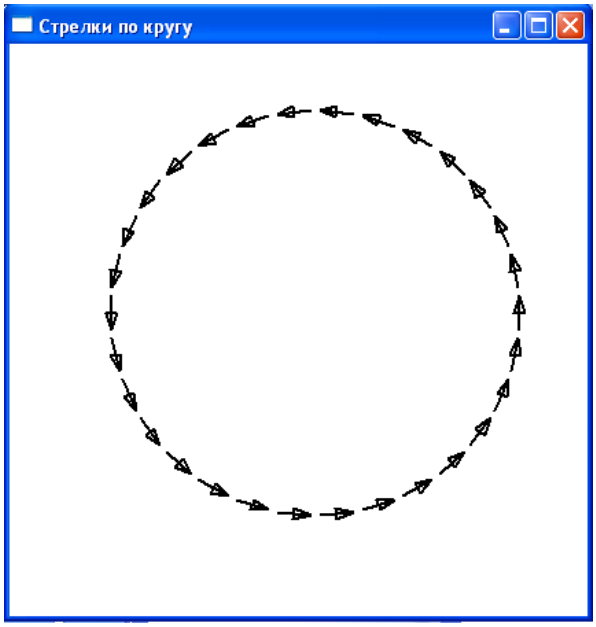


Рис.4.2.1. Поворот стрелки вокруг точки (x_0, y_0)

4.3. **Матричная запись**

Система уравнений (4.1.2) может быть записана ввиде одного матричного уравнения

$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{bmatrix} \tag{4.3.1}$$

или с использованием вектора-столбца

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{4.3.2}$$

В книгах по машинной графике запись с вектором-строкой (4.3.1) встречается чаще, чем с вектором-столбцом (4.3.2). Здесь также будет применяться запись типа (4.3.1). В такой записи i-я строка квадратной

матрицы всегда является отображением i-го единичного вектора (здесь $i = 1,2$). Вполне возможно записать в матричной форме систему уравнений (4.2.1)

$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x_0 & y_0 \end{bmatrix} + \begin{bmatrix} x - x_0 & y - y_0 \end{bmatrix} \begin{bmatrix} cos\varphi & sin\varphi \\ -sin\varphi & cos\varphi \end{bmatrix} \tag{4.3.3}$$

Однако первая часть этого уравнения не является чисто матричным произведением. В более сложных ситуациях, когда поворот совмещается с другими преобразованиями, было бы удобно иметь единое матричное произведение для каждого элементарного преобразования. На первый взгляд это кажется невозможным, если преобразование включает операцию переноса. Но с помощью матрицы преобразования 3×3 это вполне реально. Начнём с простого переноса. Пусть точка $P(x, y)$ переносится в точку $P'(x', y')$, где

$$\begin{aligned} x' &= x + a \\ y' &= y + b \end{aligned} \tag{4.3.4}$$

Эти уравнения можно переписать в виде

$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ a & b \end{bmatrix}$$

Но с учётом будущих потребностей это уравнение лучше переписать в следующей форме

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ a & b & 1 \end{bmatrix} \tag{4.3.5}$$

Легко проверить, что уравнения (4.3.4) и (4.3.5) эквивалентны. Такую запись принято называть записью в системе «*однородных координат*». Запись каждого преобразования в форме произведения матриц позволяет совмещать несколько преобразований в одном. Чтобы показать такое совмещение преобразований, объединим поворот с двумя переносами. Поворот на угол φ вокруг начала координат O был описан уравнением (4.3.1). Заменим это уравнение следующим:

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \cdot \begin{bmatrix} cos\varphi & sin\varphi & 0 \\ -sin\varphi & cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.3.6}$$

Теперь выведем новую версию уравнений (4.3.3) для описания поворота на угол φ вокруг точки (x_0, y_0); это уравнение может быть выражено формулой

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} R \tag{4.3.7}$$

через R обозначена матрица размером 3×3. Для нахождения этой матрицы R будем считать, что преобразование состоит из трёх шагов с промежуточными точками (u_1, v_1) и (u_2, v_2).

1. Преобразование для переноса точки (x_0, y_0) в начало координат O

$$\begin{bmatrix} u_1 & v_1 & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} T'$$

где

$$T' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_0 & -y_0 & 1 \end{bmatrix}$$

2. Поворот на угол φ относительно точки начала координат O

$$\begin{bmatrix} u_2 & v_2 & 1 \end{bmatrix} = \begin{bmatrix} u_1 & v_1 & 1 \end{bmatrix} R_0$$

где

$$R_0 = \begin{bmatrix} cos\varphi & sin\varphi & 0 \\ -sin\varphi & cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.3.8}$$

3. Перенос из начала координат в точку (x_0, y_0)

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} u_2 & v_2 & 1 \end{bmatrix} T$$

где

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_0 & y_0 & 1 \end{bmatrix}$$

Возможность комбинации этих шагов основана на свойстве ассоциативности матричного умножения, то есть

$$(AB)C = A(BC)$$

для любых трёх матриц A,B,C, имеющих размерности, допускающие такое умножение. Для любой части этого уравнения мы можем просто записать ABC.

Теперь найдём

$$\begin{aligned}
&\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} u_2 & v_2 & 1 \end{bmatrix} T \\
&= \left\{ \begin{bmatrix} u_1 & v_1 & 1 \end{bmatrix} R_0 \right\} T \\
&= \begin{bmatrix} u_1 & v_1 & 1 \end{bmatrix} R_0 T \\
&= \left\{ \begin{bmatrix} x & y & 1 \end{bmatrix} T' \right\} R_0 T \\
&= \begin{bmatrix} x & y & 1 \end{bmatrix} T' R_0 T \\
&= \begin{bmatrix} x & y & 1 \end{bmatrix} R
\end{aligned}$$

где

$$R = T' R_0 T$$

Это и будет искомая матрица, которая после выполнения двух матричных умножений дает

$$R = \begin{bmatrix} cos\,\varphi & sin\,\varphi & 0 \\ -sin\,\varphi & cos\,\varphi & 0 \\ c_1 & c_2 & 1 \end{bmatrix}$$

где введены обозначения

$$\begin{aligned}
c_1 &= x_0 - x_0\, cos\,\varphi + y_0\, sin\,\varphi \\
c_2 &= y_0 - x_0\, sin\,\varphi - y_0\, cos\,\varphi
\end{aligned}$$

4.4. **Проецирование и однородные координаты**

Для осознания таких понятий как проецирование и однородные координаты обратимся к некоторым общим определениям, присутствующим в разделе проективной геометрии.

Проективная геометрия – раздел геометрии, изучающий свойства фигур, не меняющиеся при проективных преобразованиях.

Проекция – является отображением множества точек в (n+1)-мерном пространстве в одну в n-мерном пространстве.

Однородные координаты обеспечивают переход одной точки n-мерного пространства к множеству точек в (n+1)-мерном пространстве по обратному закону центрального проецирования.

Рассмотрим процесс *проецирования*, который присутствует в учебниках *начертательной и проективной геометрии*, где он является основным инструментом и представлены его основные законы.

Проецирование — метод получения проекций — изображений пространственных предметов на плоскости проекций при помощи пучка воображаемых проецирующих световых или зрительных лучей [7]. При этом предмет располагается между наблюдателем и плоскостью проекций. Основным законом проецирования, который ложится в основу зрительного восприятия пространства человеком является *центральное проецирование*.

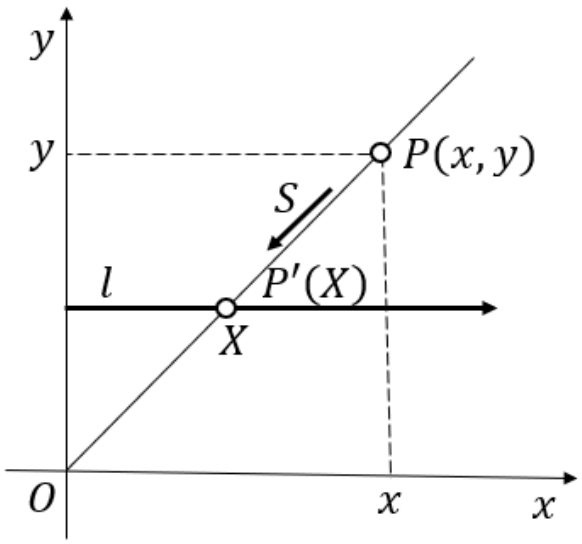


Рис.4.4.1. Пример центрального проецирования точки на прямую

На рисунке 4.4.1. показан простейший пример центрального проецирования точки $P(x, y)$ двухмерного пространства в точку $P'(X)$ на

прямой l одномерного пространства. Проецирующий луч S проходит через начало системы координат xOy (центр проекции).

В начертательной геометрии рассмотрению подлежат в основном задачи 3D-моделирования, поэтому при *центральном (коническом) проецировании* из фиксированной точки O (центра проекции) через все точки проецируемого предмета (объекта) мысленно проводят прямолинейные лучи до их пересечения с плоскостью проекций (рис.4.4.2). Точки их пересечения образуют требуемое, но увеличенное изображение предмета.

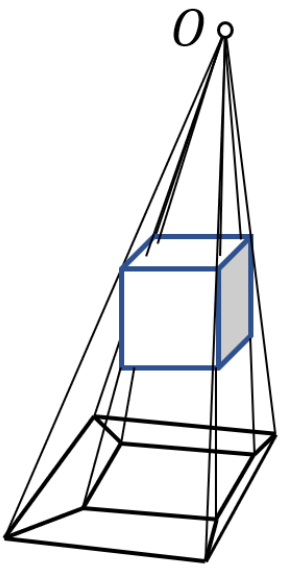


Рис.4.4.2. Центральное проецирование

Существует несколько видов закона *центрального проецирования*.

Центральная проекция широко используется в системе изображения предметов на плоскости (4.4.2). При удалении центра проекции в бесконечность конические лучи воспринимаются как параллельные.

Проецирование параллельными лучами называют *параллельным* (Рис.4.4.3).

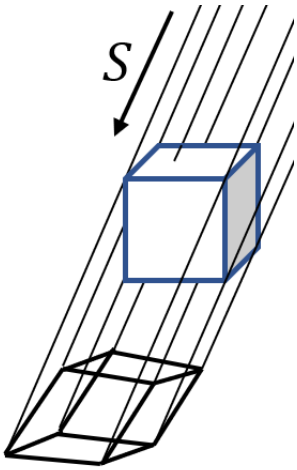


Рис.4.4.3. Параллельное проецирование

Если при этом проецирующие лучи направлены перпендикулярно плоскости проекций (рис.4.4.4), то проецирование называется *прямоугольным*, или *ортогональным.* Если эти лучи составляют с плоскостью проекций острый угол, то проецирование называется *косоугольным*. *Прямоугольное проецирование* является основным методом построения чертежей и наглядных изображений.

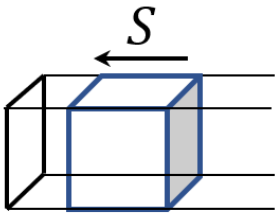


Рис.4.4.4. Прямоугольное проецирование

Перейдём к рассмотрению понятия *однородных координат.*

Промоделируем обратный процесс центральному проецированию. Для этого зададимся точкой $P(X)$ для одномерного случая. Например, равенство $X = a$ определит точку на расстоянии a от начата координат (рис.4.4.5).

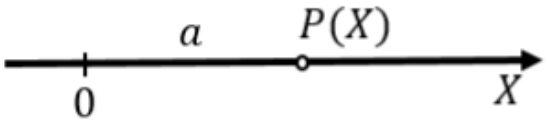


Рис.4.4.5. Задание точки в одномерном пространстве

Добавив равенству $X = a$ аргумент y получим уравнение $x = ay$. Это двухмерное уравнение прямой, проходящей через начало системы координат xOy, т.к. при $y = 0 \rightarrow x = 0$. Рассмотрим при каком аргументе y аргумент x принимает значение a. Для этого преобразуем уравнение $x = ay$ к виду $x/y = a$. Отсюда можно сделать вывод, что $x = a$ при $y = 1$. Поместим ось одномерного пространства X в систему координат xOy на единичном уровне по оси Oy и получим геометрическую интерпретацию перехода к *однородным координатам* (рис.4.4.6).

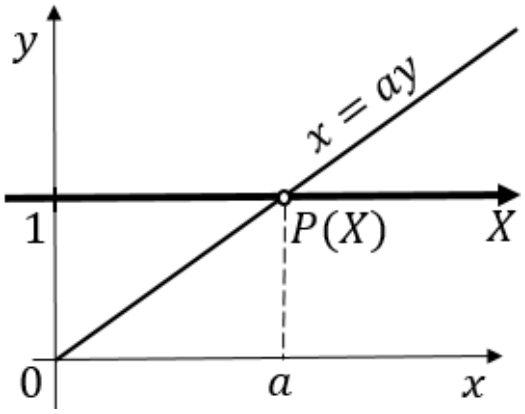


Рис.4.4.6. Геометрическая интерпретация однородных координат

Учитывая, что $X = a$ и $x/y = a$ получим выражение $X = x/y$, при $y \neq 0$ или $Xy = x,$ а значит точку X в двухмерном пространстве можно задать как (Xy, y) или $(X, 1)$ при $y = 1$.

Сравнивая рисунки 4.4.1 и 4.4.6 можно убедится, что оба случая объединены законом центрального проецирования в точке начала координат O.

Продолжим рассуждение, рассматривая по аналогии двухмерный случай перехода к трёхмерному пространству. Для этого зададимся уравнением прямой $aX + bY + c = 0$. На рисунке 4.4.7 представлена прямая в плоском пространстве XOY.

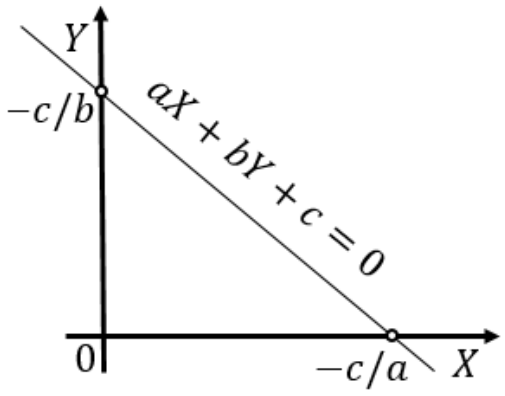


Рис.4.4.7. Прямая в плоскости XOY

По аналогии с предыдущим случаем добавим к уравнению прямой аргумент $\boldsymbol{z}$ к свободному члену $\boldsymbol{c}$ и получим уравнение плоскости $ax + by + cz = 0$. Эта плоскость в пространстве $Oxyz$ также проходит через начало координат и отсекает прямую $aX + bY + c = 0$ при пересечении с плоскостью пространства XOY, помещённого в плоскость $z = 1$ для пространства $Oxyz$ (рис.4.4.8).

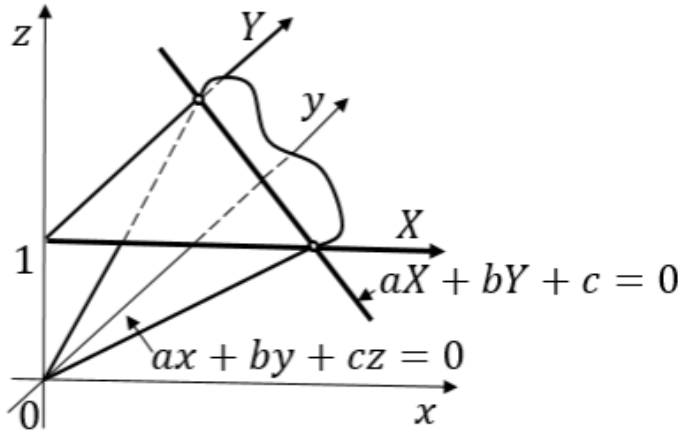


Рис.4.4.8. Переход к однородным координатам уравнения прямой

Поделим все члены уравнения $ax + by + cz = 0$ на z:

$$a\frac{x}{z} + b\frac{y}{z} + c = 0.$$

Отсюда делаем вывод, что $X = x/z,\ Y = y/z$, при $z \neq 0$, а значит точку в однородных координатах трёхмерного пространства можно записать как (Xz, Yz, z) или $(X, Y, 1)$.

В общем случае, уравнение $a_1X_1 + \cdots + a_nX_n + a_{n+1} = 0$ переходит к уравнению $a_1x_1 + \cdots + a_nx_n + a_{n+1}x_{n+1} = 0$, а значит точку n-мерного пространства $(X_1, \ldots, X_n)$ можно представить в $n+1$-мерном пространстве как $(X_1x_{n+1}, \ldots, X_nx_{n+1}, x_{n+1})$ или $(X_1, \ldots, X_n, 1)$.

Заметим также, что такой переход к увеличению размерности пространства лишает уравнение свободного коэффициента, умножая к нему новый аргумент. Это и обеспечивает *однородность* представления членов всех уравнения.

Рассмотрим систему из двух линейных уравнений, каждое из которых описывает прямую линию в двумерном пространстве:

$$\begin{cases} a_1X + b_1Y + c_1 = 0; \\ a_2X + b_2Y + c_2 = 0. \end{cases} \quad (4.4.1)$$

Если две прямые линии параллельны, то они не пересекаются и не существует пары чисел (X,Y), удовлетворяющей системе (4.4.1). Таким образом, для нахождения общей точки для прямых линий необходимо применять правило с некоторым исключением. При замене координат X и Y на однородные координаты x, y, z ситуация несколько улучшится

$$\begin{cases} a_1x + b_1y + c_1z = 0; \\ a_2x + b_2y + c_2z = 0. \end{cases} \quad (4.4.2)$$

Уравнения из системы (4.4.2) можно интерпретировать как плоскости, проходящие точку начала координат O. Эта система имеет, по крайней мере, одно тривиальное решение $x = y = z = 0$. Для геометрической интерпретации зададим для коэффициентов конкретные значения. Пусть, например, заменим систему (4.4.1) на систему

$$\begin{cases} 2X + 3Y - 6 = 0; \\ 4X + 6Y - 24 = 0. \end{cases}$$

описывающую две параллельные прямые линии, изображённые на рис.4.4.8. Тогда систему уравнений (4.4.2) можно заменить на

$$\begin{cases} 2x + 3y - 6z = 0; & (4.4.3а) \\ 4x + 6y - 24z = 0. & (4.4.3б) \end{cases}$$

В плоскости $z = 0$ эта система эквивалентна системе

$$\begin{cases} 2x + 3y = 0; \\ z = 0, \end{cases}$$

поскольку решением системы уравнения будет прямая $2x + 3y = 0$, расположенная в плоскости $z = 0$. Так что решение состоит из всех троек $(3k, -2k, 0)$, где k - любое вещественное число. В трехмерном пространстве эти точки образуют прямую линию, проходящую через точки O и $(3, -2,0)$, причём эта прямая линия представляет собой линию пересечения двух плоскостей, заданных уравнениями (4.4.3) и лежит в плоскости $z = 0$. Возвращаясь к двухмерному пространству плоскости $z = 1$, вспомним, что каждая точка (X, Y) ассоциируется с прямой линией (Xz, Yz, z) в трёхмерном пространстве. Для ненулевых значений z эта ассоциация почти тривиальна. Теперь станет понятной очень важная причина применения однородных координат. Для каждой прямой линии в двухмерном пространстве добавим один объект, называемый *бесконечно удалённой точкой*. Эта бесконечно удалённая точка не может быть обозначена в обычной системе координат, а в системе однородных координат – может. Например, бесконечно удалённая точка на прямой линии, описываемой уравнением (4.4.3.а), записывается как $(3, -2,0)$ или в виде любой тройки $(3k, -2k, 0)$ для ненулевого k. Поскольку эти тройки являются решением системы уравнений (4.4.3), то бесконечно удалённую точку можно считать точкой пересечения двух параллельных линий, изображённых на рис.4.4.8. Считается, что бесконечно удалённая точка находится в двухмерном пространстве. Каждая точка в двухмерном пространстве ассоциируется с прямой линией в трёхмерном пространстве, поэтому выясним, с какой прямой линией ассоциируется бесконечно удалённая точка $(3, -2,0)$. Поскольку эта линия должна проходить через точку начала координат $O(0,0,0)$, то искомая линия должна быть линией, проходящей через точку O и $(3, -2,0)$, то есть лежащей в плоскости $z = 0$.

Вполне резонно назвать точку $(3,-2,0)$ бесконечно удалённой точкой, поскольку её можно рассматривать как предельную точку $(3,-2,z)$ при z, стремящемся к нулю, а эта тройка в однородных координатах эквивалентна

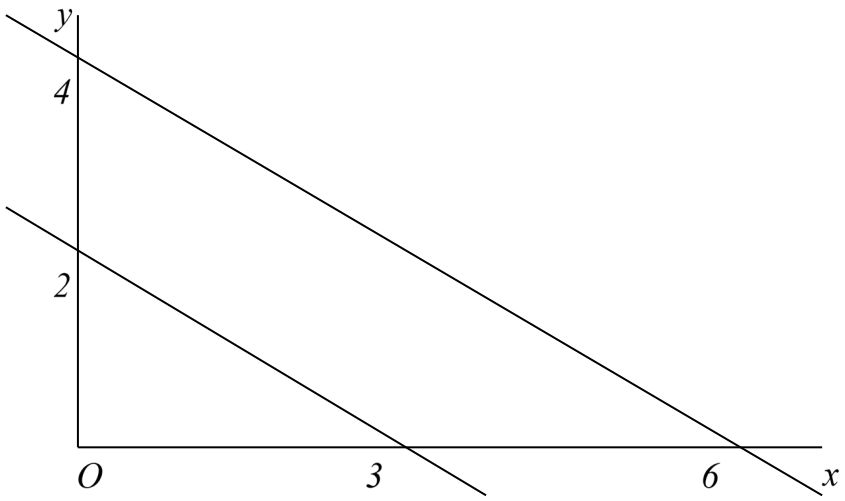


Рис.4.4.8. Параллельные прямые линии

точке $(3/z,-2/z,1)$, которая при малых значениях z стремится к бесконечности. Введение бесконечно удалённой точки позволяет утверждать, что любые две различные прямые пересекаются. Аналогично в проективной геометрии можно утверждать, что две любые различные плоскости имеют линию пересечения. Если плоскости параллельны, то все точки этой линии пересечения в однородных координатах записываются в виде $(x,y,z,0)$.

4.5. **Перенос и повороты в трёхмерном пространстве**

Если каждая точка $P(x,y,z)$ отображается на точку $P'(x',y',z')$ в соответствии с уравнениями

$$\begin{cases} x' = x + a_1 \\ y' = y + a_2 \\ z' = z + a_3 \end{cases}$$

где a_1, a_2, a_3 - константы, то это процесс называется переносом в трёхмерном пространстве. Такой перенос может быть записан в матричной форме

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} T$$

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a_1 & a_2 & a_3 & 1 \end{bmatrix}$$

При этом первая, вторая и третья строки матрицы T соответствуют отображениям бесконечно удалённых точек на координатных осях, а четвёртая строка – отображению точки $\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$. Последнее означает, что точка $\begin{bmatrix} a_1 & a_2 & a_3 & 1 \end{bmatrix}$ является отображением точки начала координат O. Поворот вокруг координатных осей может быть описан матрицей без использования однородных координат. Далее будем обращаться к однородным координатам только в тех случаях, когда они действительно необходимы.

В качестве системы координат выберем правую координатную систему, считая вращение вокруг оси положительным, если оно соответствует положительному направлению этой оси по правилу винта с правой резьбой (рис 4.5.1).

Рассмотрим поворот вокруг оси z на угол α и для сокращения обозначим $cos\alpha = c$ и $sin\alpha = s$. Тогда можно записать

$$\begin{bmatrix} x' & y' & z' \end{bmatrix} = \begin{bmatrix} x & y & z \end{bmatrix} R_z$$

$$R_z = \begin{bmatrix} c & s & 0 \\ -s & c & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

что непосредственно следует из уравнений (4.3.1). Матрицу R_z можно использовать для получения матриц R_x и R_y, определяющих поворот вокруг соответствующих осей, чисто формальным образом. Это делается путём циклических перестановок, получаемых заменой каждой из букв x, y, z на последующую, считая, что за буквой z следует буква x.

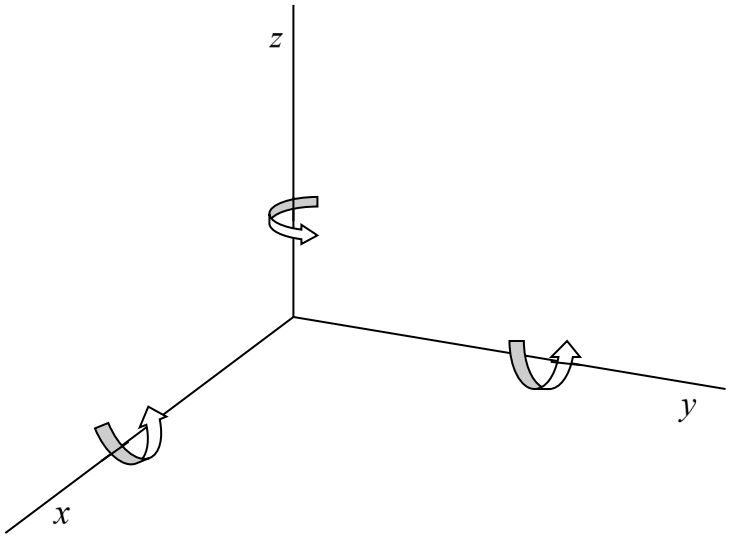


Рис.4.5.1. Вращение в положительном направлении вокруг координатных осей

Матрица R_z превратится в матрицу R_x циклическим переносом каждой строки на одну позицию и затем выполнением аналогичной операции для столбцов:

$$R_z = \begin{bmatrix} c & s & 0 \\ -s & c & 0 \\ 0 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 & 1 \\ c & s & 0 \\ -s & c & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & -s & c \\ 1 & 0 & 0 \\ 0 & c & s \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & s \\ 0 & -s & c \end{bmatrix} = R_x$$

$$R_y = \begin{bmatrix} c & 0 & -s \\ 0 & 1 & 0 \\ s & 0 & c \end{bmatrix}$$

Суммируя сказанное, получим следующие матрицы:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\,\alpha & sin\,\alpha \\ 0 & -sin\,\alpha & cos\,\alpha \end{bmatrix} \quad (4.5.1)$$

$$R_y = \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{bmatrix} \quad (4.5.2)$$

$$R_z = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (4.5.3)$$

Для поворота вокруг оси x на угол α матрица R_x используется следующим образом:

$$\begin{bmatrix} x' & y' & z' \end{bmatrix} = \begin{bmatrix} x & y & z \end{bmatrix} R_x$$

Матрицы R_y и R_z применяются аналогично.

Рассмотрим пространственное перемещение. На практике удобнее перемещать координатную систему вместо точки, но для этого требуется инвертирование матриц:

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -a_1 & -a_2 & -a_3 & 1 \end{bmatrix} \quad (4.5.4)$$

$$R_x^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\,\alpha & -sin\,\alpha \\ 0 & sin\,\alpha & cos\,\alpha \end{bmatrix} \quad (4.5.5)$$

$$R_y^{-1} = \begin{bmatrix} cos\,\alpha & 0 & sin\,\alpha \\ 0 & 1 & 0 \\ -sin\,\alpha & 0 & cos\,\alpha \end{bmatrix} \quad (1.5.6)$$

$$R_z^{-1} = \begin{bmatrix} cos\,\alpha & -sin\,\alpha & 0 \\ sin\,\alpha & cos\,\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (1.5.7)$$

Теперь можно найти матрицу R для поворота вокруг любой прямой линии, проходящей через точку начала координат O. Для определённости будем полагать, что поворот осуществляется вокруг вектора v, начало которого расположено в точке O. Тогда положительное направление вращения соответствует направлению вектора по правилу винта с правой резьбой. Как и ранее, поворот будем производить на угол α.

Если концевая точка вектора v задана в ортогональных координатах, то сначала вычислим его сферические координаты ρ, θ, φ (рис.4.5.2)

$$\rho = |v| = \sqrt{v_1^2 + v_2^2 + v_3^2}$$

Если $\rho = 0$, то будем считать, что $\theta = \varphi = 0$. В противном случае

$$\theta = \begin{cases} arctan(v_2 / v_1) & если\ v_1 > 0 \\ \pi + arctan(v_2 / v_1) & если\ v_1 < 0 \\ \pi / 2 & если\ v_1 = 0, v_2 \geq 0 \\ 3\pi / 2 & если\ v_1 = 0, v_2 < 0 \end{cases}$$

$$\varphi = arccos(v_3 / \rho)$$

Обратным вычислением будет:

$$v_1 = \rho\, sin\varphi\, cos\theta,\ v_2 = \rho\, sin\varphi\, sin\theta,\ v_3 = \rho\, cos\varphi$$

Теперь стратегия заключается в таком изменении системы координат, чтобы вектор v (ось вращения) совпадал с новым направлением положительной полуоси z. Начнем с поворота осей x, y вокруг оси z на угол θ. В соответствии с уравнением (4.5.7)

$$\begin{bmatrix} x' & y' & z' \end{bmatrix} = \begin{bmatrix} x & y & z \end{bmatrix} R_z^{-1}$$

Теперь ось x' имеет положительное направление вектора $(v_1, v_2, 0)$.

Повернём оси x', z' вокруг оси y' на угол φ до совпадения оси z'' с вектором v (рис.4.9)

Обращаясь к уравнению (4.5.6), это условие запишем как

$$\begin{bmatrix} x'' & y'' & z'' \end{bmatrix} = \begin{bmatrix} x' & y' & z' \end{bmatrix} R_y^{-1}$$

Фактический поворот вокруг вектора v на угол α теперь можно выполнить как поворот вокруг оси z''. Из уравнения (4.5.5) получим

$$\begin{bmatrix} x''' & y''' & z''' \end{bmatrix} = \begin{bmatrix} x'' & y'' & z'' \end{bmatrix} R_v$$

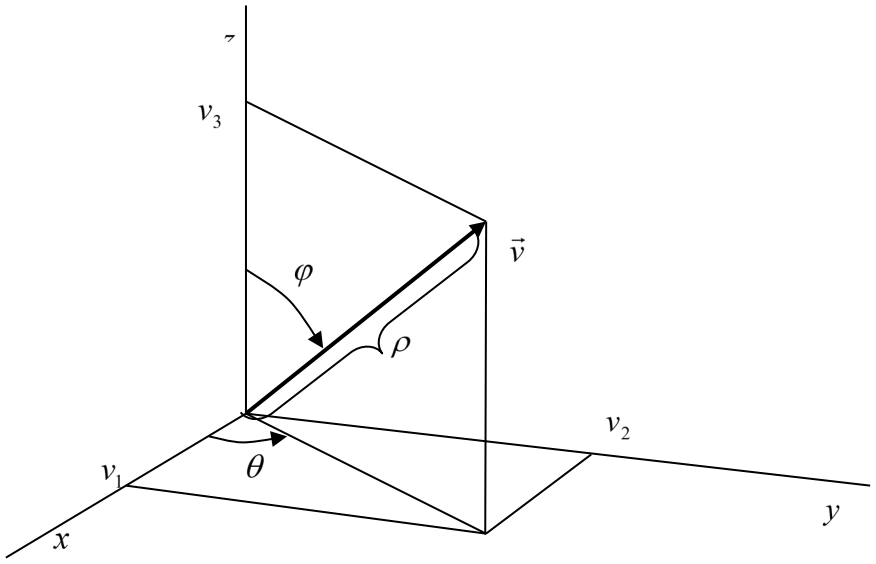


Рис4.5.2. Сферические координаты

$$R_v = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

К этому моменту достигнуто выполнение соотношения

$$\begin{bmatrix} x''' & y''' & z''' \end{bmatrix} = \begin{bmatrix} x & y & z \end{bmatrix} R_z^{-1} R_y^{-1} R_v$$

К сожалению, координаты x''', y''', z''' относятся к самой последней системе координат, тогда как их необходимо выразить в исходной системе. Обозначим эти координаты в исходной системе через x^*, y^*, z^*. Переход к исходной системе инвертированных матриц R_z^{-1} и R_y^{-1} (которые будут совпадать с матрицами R_z и R_y) в обратном порядке преобразования точки x''', y''', z''':

$$\begin{bmatrix} x^* & y^* & z^* \end{bmatrix} = \begin{bmatrix} x''' & y''' & z''' \end{bmatrix} R_y R_z$$

Это означает, что полный поворот вокруг вектора vна угол α вычисляется по следующей формуле:

$$\begin{bmatrix} x^* & y^* & z^* \end{bmatrix} = \begin{bmatrix} x''' & y''' & z''' \end{bmatrix} R_z^{-1} R_y^{-1} R_v R_y R_z$$

где

$$R_z^{-1} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_y^{-1} = \begin{bmatrix} \cos\varphi & 0 & \sin\varphi \\ 0 & 1 & 0 \\ -\sin\varphi & 0 & \cos\varphi \end{bmatrix}$$

$$R_v = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_y = \begin{bmatrix} \cos\varphi & 0 & -\sin\varphi \\ 0 & 1 & 0 \\ \sin\varphi & 0 & \cos\varphi \end{bmatrix}$$

$$R_z = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Для последующего применения запишем

$$R_z^{-1} R_y^{-1} R_v R_y R_z = R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{4.5.8}$$

До сих пор обсуждалось решение задачи о повороте относительно вектора, привязанного к точке начала системы координат O. Теперь нужно устранить это последнее ограничение и поставить задачу определения поворота относительно вектора, начало координат которого расположено в любой произвольной точке $A(a_1, a_2, a_3)$.

Для этого будем использовать вектор v для вычисления матрицы R в уравнении (4.5.8) таким же образом, как и ранее. Затем нужно выполнить три следующих шага:

1. Обращаясь к уравнению (4.5.4), выполним перенос из заданной точки в точку начала координат O, используя однородные координаты и следующую матрицу:

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -a_1 & -a_2 & -a_3 & 1 \end{bmatrix}$$

2. Теперь можем осуществить поворот относительно оси, проходящей через O, как и ранее, но матрицу R из уравнения (4.5.8) необходимо расширить тривиальным образом, чтобы можно было использовать однородные координаты

$$R* = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3. Применить преобразование, обратное шагу 1, используя матрицу

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a_1 & a_2 & a_3 & 1 \end{bmatrix}$$

После этого матрица обобщённого поворота вычисляется как

$$R_{GEN} = T^{-1} R * T$$

и её можно использовать следующим образом:

$$[x^* \quad y^* \quad z^* \quad 1] = [x \quad y \quad z \quad 1] R_{GEN}.$$

5. ПЕРСПЕКТИВНЫЕ ИЗОБРАЖЕНИЯ

При необходимости получения перспективной проекции задаётся большое количество точек $P(x,y,z)$, принадлежащих объекту, для которых предстоит вычислить координаты точек изображения $P'(X,Y)$ на картинке. Для этого нужно только преобразовать координаты точки P из так называемых *мировых координат* (x,y,z) в *экранные координаты* (X,Y) её центральной проекции P'. Будем предполагать, что экран расположен между объектом и глазом E. Для каждой точки P объекта прямая линия PE пересекает экран в точке P'. Это отображение удобно выполнять в два этапа. Первый этап будем называть *видовым преобразованием* – точка P остаётся на своём месте, но система мировых координат переходит в систему *видовых координат*. Второй этап называется *перспективным преобразованием*. Это точное преобразование точки P в точку P', объединённое с переходом из системы трёхмерных видовых координат в систему двухмерных экранных координат:

5.1. Видовое преобразование

Для выполнения видовых преобразований должны быть заданы точка наблюдения, совпадающая с глазом, и объект. Желательно, чтобы система координат была правой. Будет удобно, если начало её координат располагается где-то вблизи центра объекта, поскольку объект наблюдается в направлении от E к O. Предположим, что это условие выполняется. Пусть точка наблюдения E будет задана в сферических координатах ρ, θ, φ по отношению к мировым координатам. То есть мировые координаты могут быть вычислены по формулам:

$$\begin{aligned} x_E &= \rho \sin\varphi \cos\theta \\ y_E &= \rho \sin\varphi \sin\theta \\ z_E &= \rho \cos\varphi \end{aligned} \qquad (5.1.1)$$

Обозначение сферических координат схематически изображены на рис. 5.1.1. Говорят, что вектор направления EO (равный $-OE$) определяет направление наблюдения. Из точки наблюдения E можно видеть точки объекта только внутри некоторого конуса, ось которого совпадает с линией EO, а вершина – с точкой E. Если заданы ортогональные координаты x_E, y_E, z_E точки E, то можно вычислить её сферические координаты (п.4.5).

Конечной задачей является вычисление экранных координат X, Y, для которых оси X и Y лежат в плоскости экрана, расположенной между точками E и O и перпендикулярной направлению наблюдения EO. Начало системы видовых координат располагается в точке наблюдения E (рис.5.1.2). При направлении взгляда из E в O положительная полуось x_e направлена вправо, а положительная полуось y_e - вверх. Такое направление осей позволит впоследствии определить экранные оси в тех же направлениях. Направление оси z_e выбирается таким образом, что значения координат увеличиваются по мере удаления от точки наблюдения.

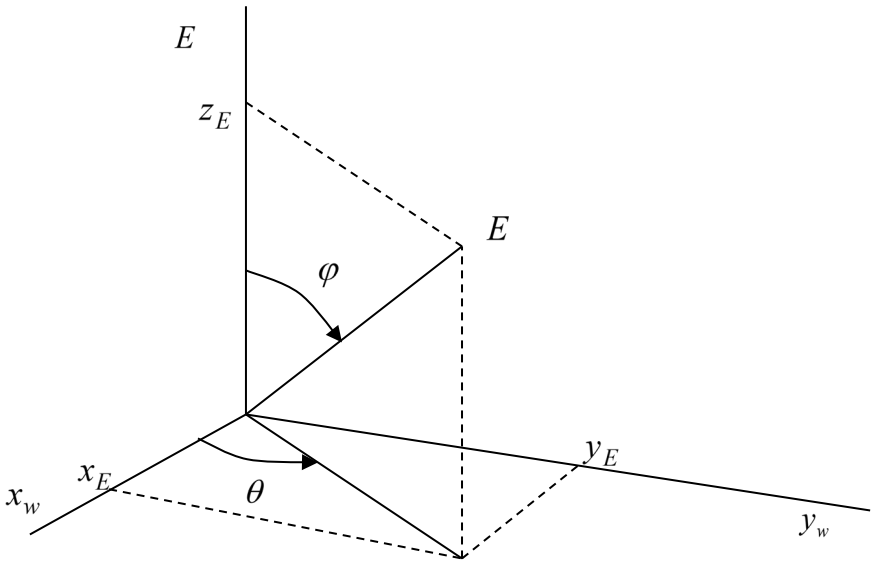


Рис.5.1.1. Сферические координаты точки наблюдения

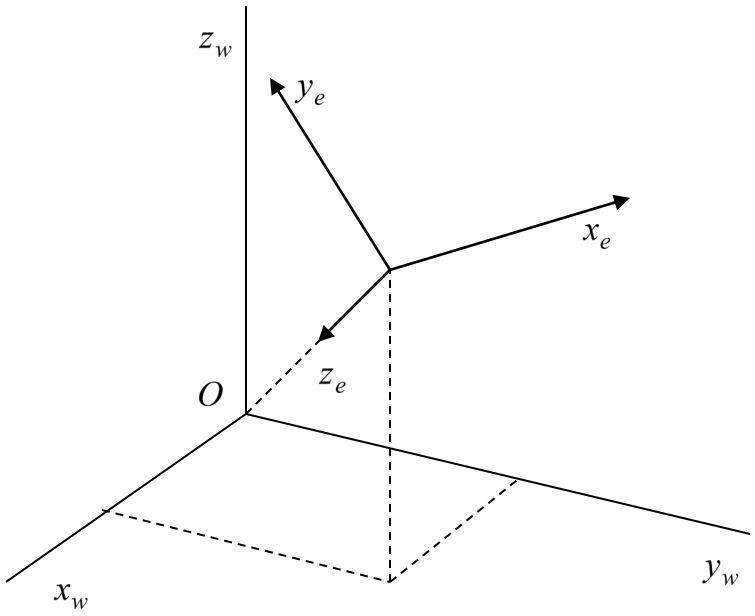


Рис.5.1.2 Система видовых координат

Видовое преобразование может быть записано в форме

$$\begin{bmatrix} x_e & y_e & z_e & 1 \end{bmatrix} = \begin{bmatrix} x_w & y_w & z_w & 1 \end{bmatrix} V \qquad (5.1.2)$$

где V - матрица видового преобразования размерностью 4×4. Для нахождения матрицы V предположим, что преобразования отображения могут быть составлены из четырёх элементарных преобразований, для которых легко написать свои матрицы преобразований. Матрица V получается путём перемножения этих четырёх матриц. Фактически каждое из

четырёх преобразований изменяет координаты и, следовательно, определяется матрицей, обратной матрице, соответствующей преобразованию точки.

Перенос начала из O в E

Выполним такой перенос системы координат, при котором точка E становится новым началом координат. Матрица такого изменения координат выглядит так:

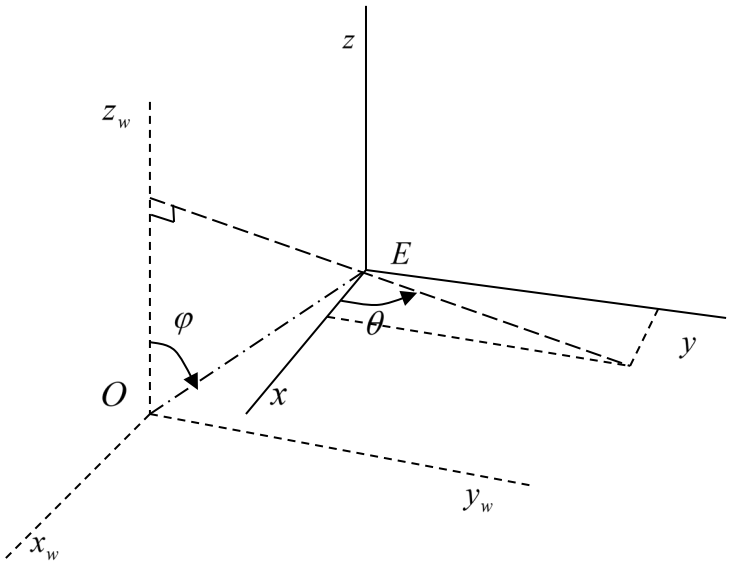


Рис.5.1.3. Новые оси после переноса

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_E & -y_E & -z_E & 1 \end{bmatrix} \qquad (2.1.3)$$

Новая система координат показана на рис. 5.1.3.

Поворот координатной системы вокруг оси z

Обращаясь к рисунку 5.1.3, повернём систему координат вокруг оси z на угол $\pi/2-\theta$ в отрицательном направлении. В результате ось Oy совпадает по направлению с горизонтальной составляющей отрезка OE, а ось Ox будет расположена перпендикулярно отрезку OE. Матрица такого изменения

координат будет совпадать с матрицей для поворота точки на такой же угол в положительном направлении. Матрица 3×3 для этого поворота равна

$$R_z = \begin{bmatrix} cos(\pi/2-\theta) & sin(\pi/2-\theta) & 0 \\ -sin(\pi/2-\theta) & cos(\pi/2-\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} sin\,\theta & cos\,\theta & 0 \\ -cos\,\theta & sin\,\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Новое положение осей показано на рис. 5.1.4.

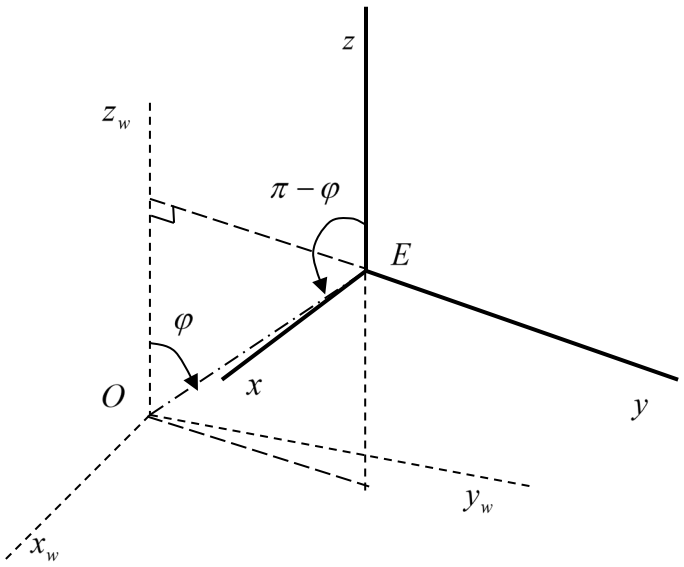


Рис.5.1.4. Новые оси после поворота вокруг оси Oz

Поворот системы координат вокруг оси *Ox*

Поскольку новая ось *Oz* должна совпадать по направлению с отрезком *EO*, повернём систему координат вокруг оси *Ox* на угол $\pi-\varphi$ в положительном направлении, что соответствует повороту точки на угол $-(\pi-\varphi)=\varphi-\pi$. Получаем матрицу:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\varphi-\pi) & sin(\varphi-\pi) \\ 0 & -sin(\varphi-\pi) & \cos(\varphi-\pi) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -\cos(\varphi) & -sin(\varphi) \\ 0 & sin(\varphi) & -\cos(\varphi) \end{bmatrix} \quad (2.1.4)$$

Изменение направления оси *Ox*

Оси *Oy* и *Oz* имеют правильную ориентацию, а ось *Ox* должна быть направлена в противоположную сторону. Поэтому необходима матрица для выполнения преобразования $x' = -x$, то есть

$$M_{yz} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

После этого завершающего преобразования получим систему видовых координат, изображённую на рис. 5.1.2.

Вычислим матрицу отображения V как матричное произведение

$$V = TR_z^* R_x^* M_{yz}^*$$

где обозначение R^* использовано для матрицы 4×4, полученной путём приведения матрицы R, имеющей размерность 3×3, к однородным координатам. Матричное произведение не коммутативно (т.е. $AB \neq BA$), но ассоциативно, поэтому

$$V = T(R_z R_x M_{yz})^*$$

Таким образом, можно работать с матрицами 3×3 до тех пор, пока это возможно.

Заменим обозначениями выражения:

$$\cos\varphi = a \quad \cos\theta = c$$
$$\sin\varphi = b \quad \sin\theta = d$$

откуда следует, что $a^2 + b^2 = 1$ и $c^2 + d^2 = 1$.

На основе уравнения 5.1.1, перепишем уравнение переноса системы координат в виде:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\rho bc & -\rho bd & -\rho a & 1 \end{bmatrix},$$

а уравнения поворотов как:

$$R_z = \begin{bmatrix} d & c & 0 \\ -c & d & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -a & -b \\ 0 & b & -a \end{bmatrix} \quad R_z R_x = \begin{bmatrix} d & -ac & -bc \\ -c & -ad & -bd \\ 0 & b & -a \end{bmatrix}$$

Умножив эту матрицу справа на матрицу M_{yz} получим:

$$R_z R_x M_{yz} = \begin{bmatrix} -d & -ac & -bc \\ c & -ad & -bd \\ 0 & b & -a \end{bmatrix}.$$

Затем искомую матрицу отображения V найдем как произведение двух матриц: T и $(R_z R_x M_{yz})^*$, откуда

$$V = \begin{bmatrix} -d & -ac & -bc & 0 \\ c & -ad & -bd & 0 \\ 0 & b & -a & 0 \\ v_{41} & v_{42} & v_{43} & 1 \end{bmatrix},$$

где

$$v_{41} = \rho bcd - \rho bcd = 0$$
$$v_{42} = \rho abc^2 + \rho abd^2 - \rho ab = \rho\{ab(c^2 + d^2) - ab\} = \rho(ab - ab) = 0$$
$$v_{43} = \rho b^2 c^2 + \rho b^2 d^2 + \rho a^2 = \rho\{b^2(c^2 + d^2) + a^2\} = \rho(b^2 + a^2) = \rho$$

Таким образом, находим:

$$V = \begin{bmatrix} -\sin\theta & -\cos\varphi\cos\theta & -\sin\varphi\cos\theta & 0 \\ \cos\theta & -\cos\varphi\sin\theta & -\sin\varphi\sin\theta & 0 \\ 0 & \sin\varphi & -\cos\varphi & 0 \\ 0 & 0 & \rho & 1 \end{bmatrix}$$

Запишем уравнение видового преобразования, выраженное через сферические координаты (ρ, θ, φ):

$$[x_e \quad y_e \quad z_e \quad 1] = [x \quad y \quad z \quad 1]V.$$

Уже сейчас можно использовать видовые координаты (x_e, y_e), просто игнорируя координату z_e. В этом случае будет получена так называемая *ортогональная проекция.* Эту проекцию можно также считать перспективной, полученной при удалении точки наблюдателя в бесконечность.

5.2. **Перспективное преобразование**

Далее мировые координаты уже не будут затрагиваться. Поэтому видовые координаты будут обозначаться просто (x, y, z) вместо (x_e, y_e, z_e).

На рисунке 5.2.1 выбрана точка Q, видовые координаты которой равны $(0,0,d)$ для некоторого положительного числа d. Плоскость $z = d$ определяет экран, который будем использовать. Таким образом, *экран* – это плоскость, проходящая через точку Q и перпендикулярная оси z. Экранные координаты определяются привязкой начала к точке Q, а оси X и Y имеют такие же направления, как оси Ex и Ey соответственно. Для каждой точки объекта P точка изображения P' определяется как точка пересечения прямой линии PE и экрана. Рассмотрим точку P с нулевой y - координатой. Треугольники EPR и $EP'Q$ подобны. Следовательно, $\frac{P'Q}{EQ} = \frac{PR}{ER}$. Отсюда будем иметь $\frac{X}{d} = \frac{x}{z}$ или $X = d \cdot \frac{x}{z}$.

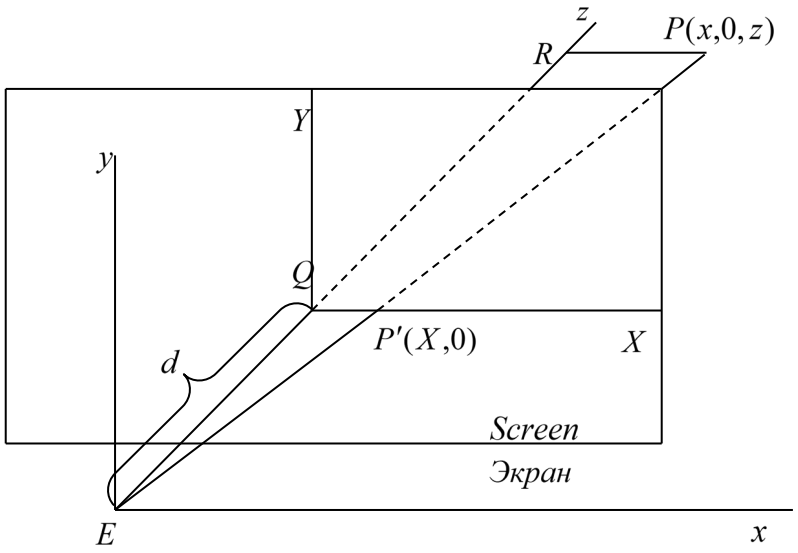


Рис.5.2.1. Экран и видовые координаты

Аналогично можем получить $Y = d \cdot \frac{y}{z}$.

Поскольку ось Oz видовой системы координат совпадает с прямой линией EO, которая пересекает экран в точке Q, то начало системы экранных координат будет находиться в центре изображения. Для переноса начала координат в левый нижний угол с размерами экрана $2c_1 \times 2c_2$ можно применить уравнения $X = d \cdot \frac{x}{z} + c_1 \quad Y = d \cdot \frac{y}{z} + c_2$.

Исходное расстояние d, помещающее весь объект в габариты картинки можно определить из отношения:

$$d = \rho \cdot \frac{\textit{размер картинки}}{\textit{размер объекта}}.$$

Рассмотрим общую концепцию перспективного сокращения.

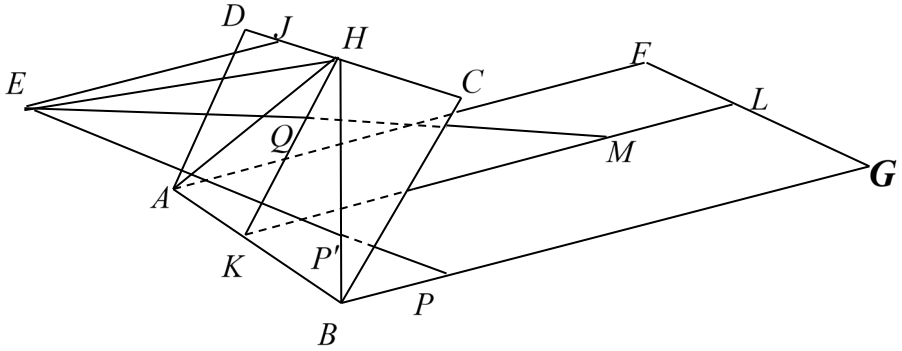


Рис.5.2.2. Точка схода H

На рисунке 5.2.2 изображены точки наблюдения E и экрана $ABCD$. Взгляд направлен из точки E в точку Q. Прямые линии AF, BG, EH параллельны, и будем считать их горизонтальными. Представим плоскость, проходящую параллельные линии EH и BG. Эта плоскость пересекает плоскость экрана по прямой линии BH. Таким образом, каждая точка P на прямой линии BG будет иметь свою центральную проекцию P', лежащую на прямой BH, при условии, что точка E является центром проекции. Пусть точка P удаляется от точки B в бесконечность, тогда её проекция P' будет приближаться к точке H. Это означает, что H - точка схода для прямой линии, проходящей через точки B и G. В терминах проективной геометрии точка H представляет собой проекцию бесконечно удалённой точки, лежащей на прямой BG.

6. АППРОКСИМАЦИЯ

6.1. Непараметрические кривые

Математически кривая может быть представлена в параметрической или непараметрической форме. Непараметрическая кривая задается в виде явной функции (например, $z = f(x, y)$) или неявной ($f(x, y, z) = 0$). Для плоской кривой явное непараметрическое представление записывается как:

$$y = f(x).$$

Например, для уравнения $y = kx + b$ каждому аргументу x соответствует единственный y. Это означает, что уравнение окружности уже не может быть описано таким образом. На помощь приходит неявная форма записи $f(x, y) = 0$. Т.е. можно записать уравнение $r^2 - x^2 - y^2 = 0$.

Общий вид неявного уравнения второй степени

$$ax^2 + 2bxy + cy^2 + 2dx + 2ey + f = 0$$

позволяет описывать различные двумерные квадратичные и линейные кривые, называемые коническими сечениями (Рис.6.1.1). Определяя коэффициенты *a*, *b*, *c*, *d*, *e* и *f*, получаем разные конические сечения. Если сечение задано относительно локальной системы координат и проходит через ее начало, то $f = 0$. Для того чтобы провести кривую через данные точки, используются граничные условия.

Пусть $c = 1.$, тогда сегмент кривой между двумя точками определяется пятью независимыми условиями, из которых вычисляются оставшиеся коэффициенты *a*, *b*, *d*, *e* и *f*. Например, можно указать положение крайних точек, значения первой производной в них и промежуточную точку на кривой.

Если $b = 0.$ и $c = 1.$, то аналитическое представление кривой получается с помощью только четырех дополнительных условий, например положения концевых точек и значения первой производной в них. Кривая при $a = 1.$, $b = 0.$ и $c = 1.$ еще проще:

$$x^2 + y^2 + 2dx + 2ey + f = 0.$$

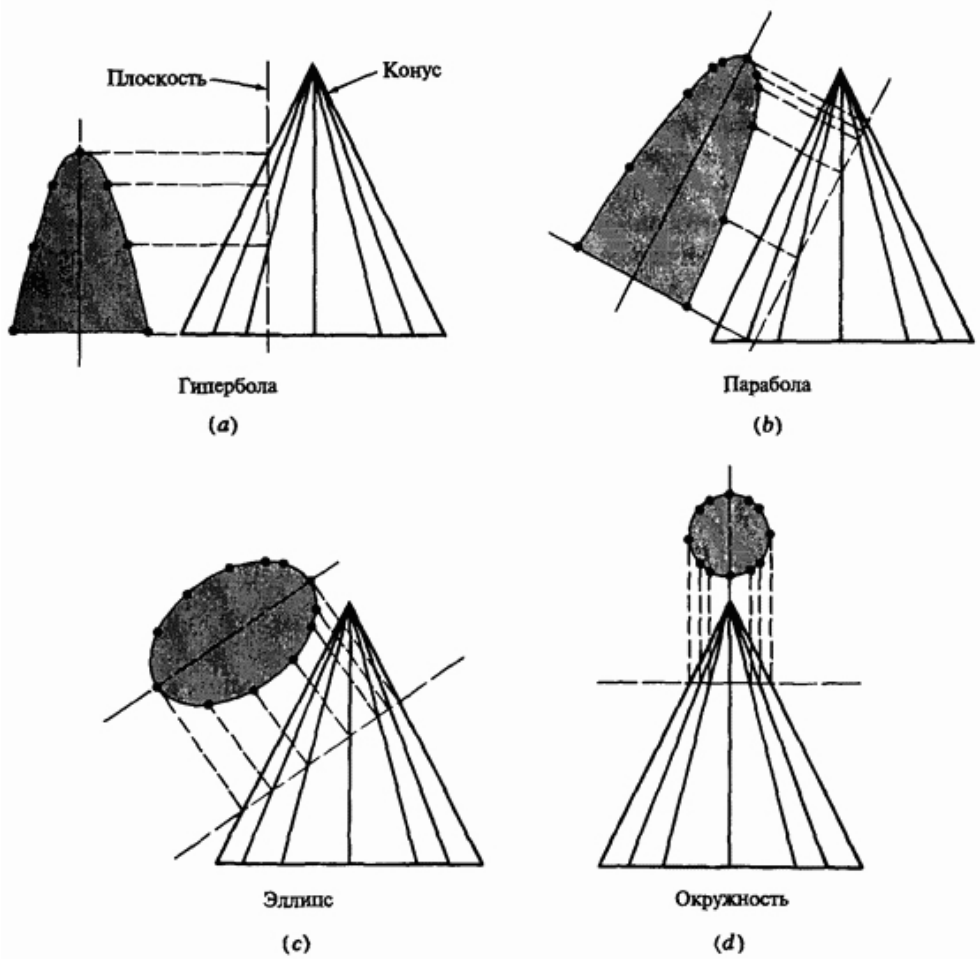


Рис.6.1.1. Основные виды конических сечений

Условиями для вычисления остальных коэффициентов могут служить концевые точки и первая производная в одной из них, или третья точка на кривой.

Прямая линия получается при $a = b = c = 0$. Уравнение принимает вид $dx + ey + f = 0$.

6.2. Параметрические кривые

В параметрическом виде каждая координата точки кривой представлена как функция одного параметра. Значение параметра задает координатный вектор точки на кривой. Для двумерной кривой с параметром *t* координаты точки равны:

$$x = x(t), \;\; y = y(t).$$

Векторное представление точки на кривой:

$$P(t) = [x(t), y(t)].$$

Самое простое параметрическое представление у прямой. Для двух векторов положения P_1 и P_2 параметрический вид отрезка прямой между ними такой:

$$P(t) = P_1 + (P_2 - P_1)t, \quad 0 \le t \le 1.$$

Поскольку $P(t)$ – вектор, то для каждой его компоненты существует параметрическое представление

$$x(t) = x_1 + (x_2 - x_1)t,$$
$$y(t) = y_1 + (y_2 - y_1)t.$$

Рассмотрим пример регулярной аппроксимации окружности в первом квадранте для непараметрического и параметрического вида. Непараметрический вид можно представить как

$$y = +\sqrt{1 - x^2}, \ 0 \le x \le 1$$

На рисунке 6.2.1 демонстрируется три подхода. Рисунок 6.2.1.a моделирует неявную форму представления, где наблюдается явное искажение аппроксимации по сравнению с остальными случаями. Точки на дуге соответствуют равным приращениям x. При этом дуга состоит из отрезков разной длины, и получается весьма приблизительное графическое представление окружности. Кроме того, расчет квадратного корня - вычислительно дорогостоящая операция.

Рисунок 6.2.1.b параметром выступает угол поворота θ. Стандартная параметрическая форма единичной окружности:

$$x = cos\theta, \quad y = sin\theta, \qquad 0 \le \theta \le 2\pi,$$

или $$P(\theta) = [x \quad y] = [cos\theta \quad sin\theta].$$

Параметрическое представление кривой не единственно, например,

$$P(t) = \left[\frac{(1 - t^2)}{(1 + t^2)} \quad \frac{2t}{(1 + t^2)}\right], \ 0 \le t \le 1$$

На рисунке 6.2.2 демонстрируется взаимосвязь параметров

$$x = cos\theta = \frac{1 - t^2}{1 + t^2}, \qquad y = sin\theta = \frac{2t}{1 + t^2}$$

$$r^2 = x^2 + y^2 = \left(\frac{1-t^2}{1+t^2}\right)^2 + \left(\frac{2t}{1+t^2}\right)^2 = \frac{1-2t^2+t^4+4t^2}{(1+t^2)^2} = \frac{(1+t^2)}{(1+t^2)^2} = 1,$$

где r – единичный радиус.

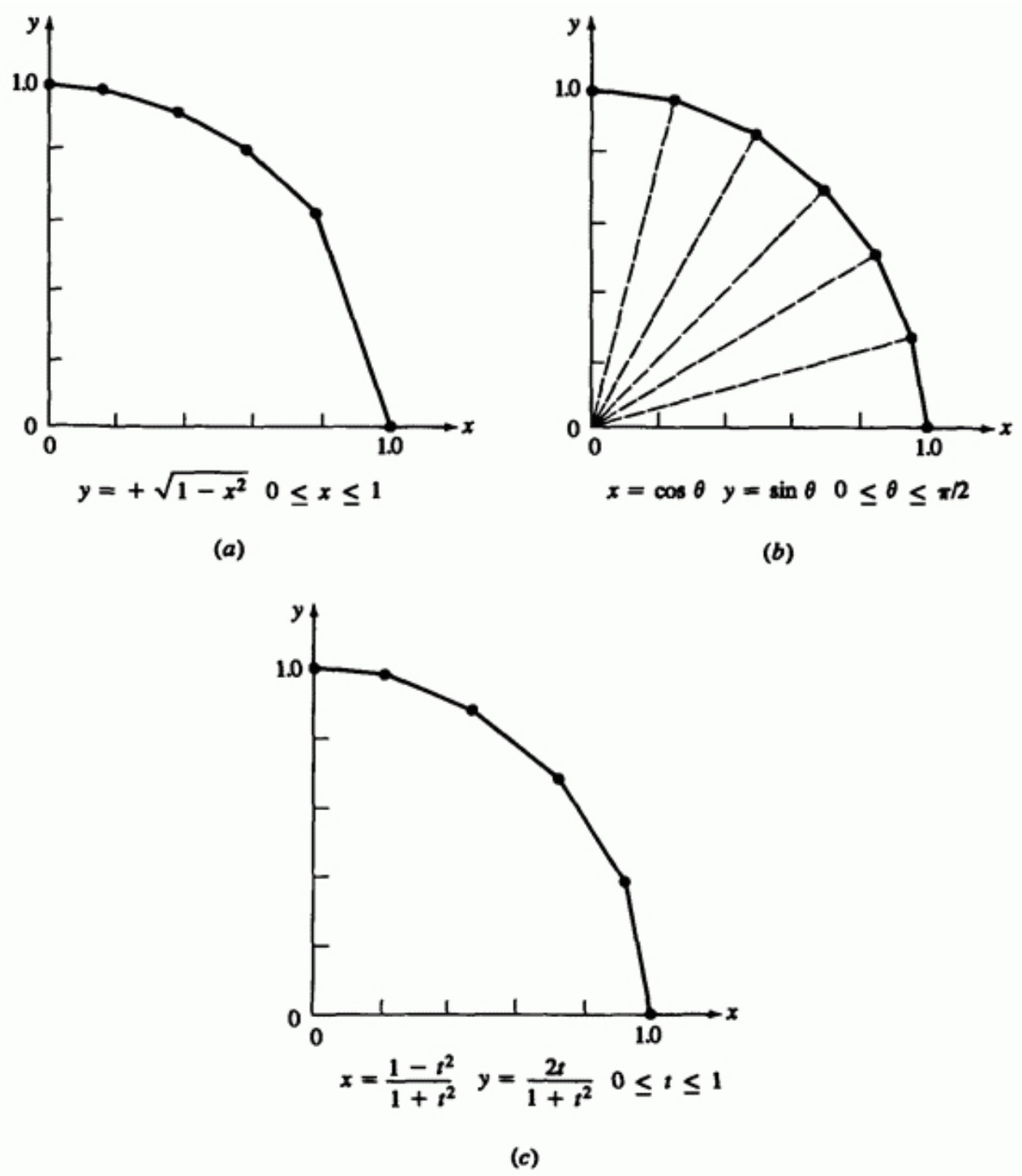


Рис.6.2.1 Представление окружности для первого квадранта

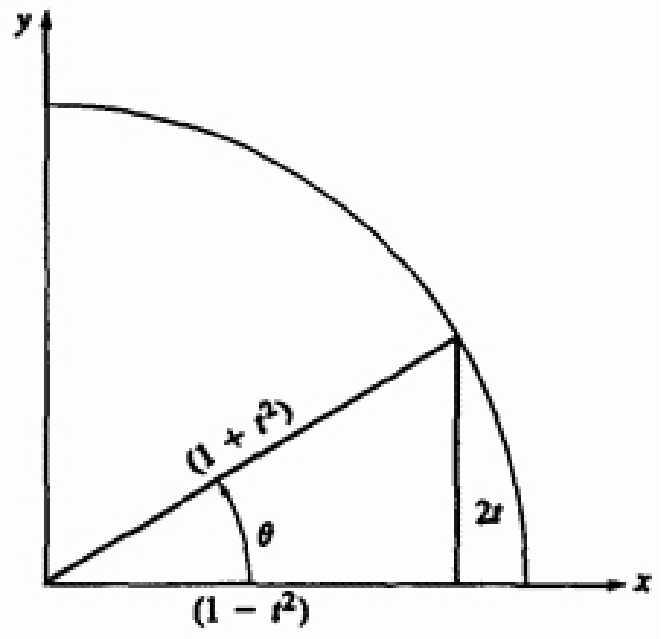


Рис.6.2.2 Связь между параметрическими представлениями

6.3. Параметрическое представление окружности

Параметрическое представление окружности радиуса r с центром в начале координат:

$$x = rcos\theta, \quad y = sin\theta, \ 0 \le \theta \le 2\pi. \qquad (6.3.1)$$

где θ - параметр. Равномерное приращение дает приличное изображение, но алгоритм неэффективен из-за частого вызова тригонометрических функций. Рассмотрим более быстрый алгоритм.

Полной окружности соответствует диапазон изменения параметра θ от 0 до 2π. Если рассматривать некоторое фиксированное число равномерно распределенных точек по окружности, то приращение параметра между точками можно считать константой. Координаты любой точки на окружности с центром в начале координат

$$x_{i+1} = rcos(\theta_i + \Delta\theta), \qquad y_{i+1} = rcos(\theta_i + \Delta\theta)$$

где θ_i - значение параметра для точки x_i, y_i.

По формуле суммы углов имеем

$$x_{i+1} = r(cos\theta_i cos\Delta\theta - sin\theta_i sin\Delta\theta), y_{i+1} = r(cos\theta_i sin\Delta\theta - sin\theta_i cos\Delta\theta)$$

Применим уравнение (6.3.1) для $\theta = \theta_i$

$$x_i = rcos\theta_i,\ \ y_i = rsin\theta_i.$$

получим рекурсивные уравнения

$$x_{i+1} = x_i cos\Delta\theta - y_i sin\Delta\theta,\ \ y_{i+1} = x_i sin\Delta\theta + y_i cos\Delta\theta \qquad (6.3.2)$$

это соответствует повороту точки x_i, y_i на $\Delta\theta$.

Так как $\Delta\theta$ постоянно и равно $2\pi/(n-1)$, где n - количество равномерно распределенных по окружности точек, значения синуса и косинуса достаточно вычислить только один раз. Во внутреннем цикле используются только четыре умножения, вычитание и сложение, поэтому алгоритм работает очень быстро. Результат на рис. 6.3.1 соответствует расчетам по формуле (6.3.1).

Окружность с центром в произвольной точке получается переносом окружности соответствующего радиуса с центром в начале координат. В некоторых случаях можно упростить задачу: сначала строить единичную окружность с центром в начале координат, а затем, комбинируя перенос и масштабирование, получить окружности с любым радиусом и центром.

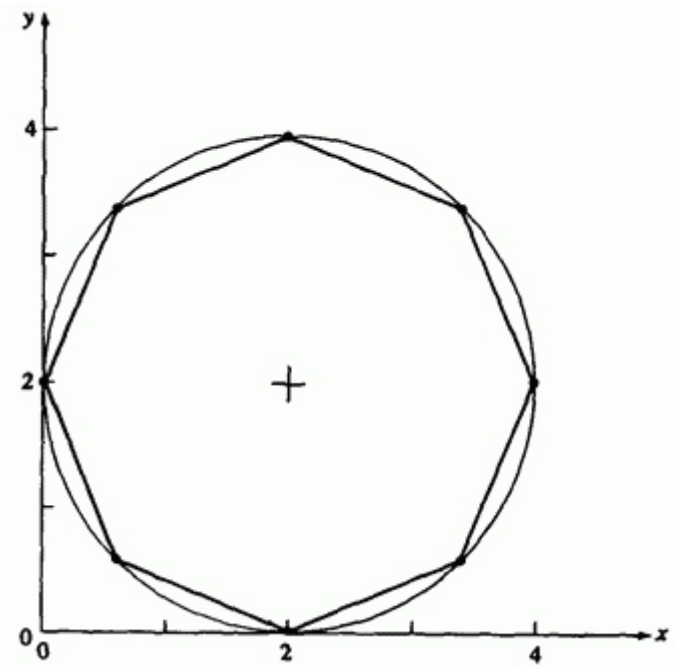


Рис. 4.6.1 Единичная параметрическая окружность с n=8.

Пример 1 Генерация параметрической окружности

Построить окружность радиуса 2 с центром в точке (2, 2). Рассмотрим два метода. *Первый:* построить окружность радиуса 2 с центром в начале координат и перенести на 2 единицы по x и y . *Второй:* построить единичную

окружность с центром в начале координат, увеличить в два раза и, наконец, перенести. Воспользуемся вторым методом.

Пусть для простоты на окружности лежат восемь точек, хотя обычно требуется гораздо большее количество. На самом деле количество точек зависит от радиуса.

Окружность - это замкнутая кривая, поэтому первая ($\theta = 0$) и последняя ($\theta = 2\pi$) точки совпадают. Следовательно, чтобы получить n различных точек на окружности, нужно вычислить $n + 1$ точку. Для незамкнутых кривых это необязательно.

Найдем $\Delta\theta$:

$$\Delta\theta = \frac{2\pi}{(n+1-1)} = \frac{2\pi}{n} = \frac{2\pi}{8} = \frac{\pi}{4}.$$

Исходные значения x и y из уравнения (6.3.1) при $\theta = 0$ таковы:

$$x_1 = rcos\theta_1 = (1)\cos(0) = 1,$$

$$y_1 = rsin\theta_1 = (1)\sin(0) = 0.$$

Теперь по формуле (6.3.2) получим остальные семь точек. Для первой

$$sin\Delta\theta = sin\frac{\pi}{4} = \frac{\sqrt{2}}{2},\ \ cos\Delta\theta = cos\frac{\pi}{4} = \frac{\sqrt{2}}{2}.$$

$$x_2 = x_1 cos\Delta\theta - y_1 sin\Delta\theta = (1)\left(\frac{\sqrt{2}}{2}\right) - (0)\left(\frac{\sqrt{2}}{2}\right) = \left(\frac{\sqrt{2}}{2}\right),$$

$$y_2 = x_1 sin\Delta\theta + y_1 cos\Delta\theta = (1)\left(\frac{\sqrt{2}}{2}\right) + (0)\left(\frac{\sqrt{2}}{2}\right) = \left(\frac{\sqrt{2}}{2}\right).$$

Результаты для остальных точек собраны в таблице 6.3.1.

Применив двумерные преобразования, определим преобразование, которое масштабирует полученные результаты с коэффициентом 2 и переносит центр окружности в точку (2, 2):

$$[S] = [M][T] = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 2 & 2 & 1 \end{bmatrix}.$$

.

Применяя это преобразование к точке (*x, y*), получаем

$$[x_1 \quad y_1 \quad 1][T] = [1 \quad 0 \quad 1]\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 2 & 2 & 1 \end{bmatrix} = [4 \quad 2 \quad 1]$$

как и ожидалось. Полный результат приведен в таблице 6.3.2 и на рис. 6.3.1.

Таблица 6.3.1 Результаты для единичной окружности

i	x_i	y_i
1	1	0
2	$\sqrt{2}/2$	$\sqrt{2}/2$
3	0	1
4	$-\sqrt{2}/2$	$\sqrt{2}/2$
5	-1	0
6	$-\sqrt{2}/2$	$-\sqrt{2}/2$
7	0	-1
8	$\sqrt{2}/2$	$-\sqrt{2}/2$
9	1	0

Таблица 6.3.2 Результаты для окружности радиуса 2 с центром в (2, 2)

i	x_i	y_i
1	4	2
2	3,414	3,414
3	2	4
4	0,586	3,414
5	0	2
6	0,586	0,586
7	2	0
8	3,414	0,586
9	4	2

Ограничивая диапазон параметра θ, можно получить дуги окружности. Например, $0 \le \theta \le 2\pi$ соответствует четверти круга в первом квадранте, а $\pi \le \theta \le 3\pi/2$ - четверти круга в третьем квадранте.

6.4 Параметрическое представление эллипса

Достаточно хорошее представление окружности получается, если соединить отрезками некоторое количество равномерно распределенных на ней точек. Однако, если рассчитывать точки эллипса через равные приращения угла, изображение будет неверным, как показано штриховой линией на рис. 6.4.1. Особенно сильно неточности проявляются на концах, где кривизна слишком велика и требуется большее количество точек.

Другой метод основан на равных приращениях по периметру и дает хороший результат для достаточно большого количества точек. Недостатки его в том, что указывается слишком много точек на сторонах с малой кривизной, и вычисление равных частей периметра требует $dy = bcos\theta d\theta$ сложного расчета эллиптического интеграла. Что нам необходимо, это малые приращения параметра у концов, где кривизна велика, и большие приращения параметра вдоль сторон с малой кривизной.

Такое распределение точек получается из параметрического представления эллипса с центром в начале координат, большой полуосью a и малой полуосью b :

$$x = acos\theta,\ y = bsin\theta. \qquad (6.4.1)$$

где θ - параметр.

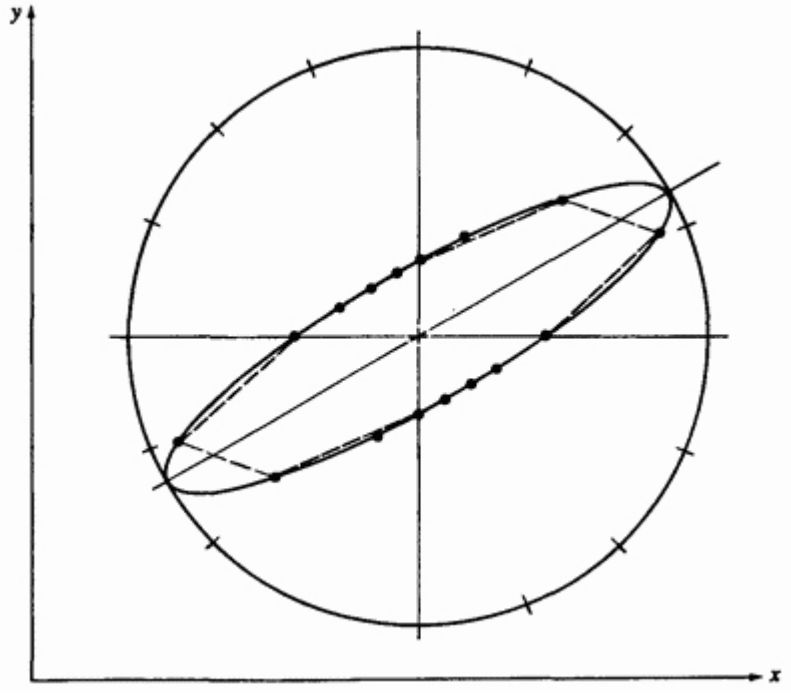


Рис. 6.4.1 Равноугольное представление сильно вытянутого эллипса.

Диапазон от 0 до 2π задает полный эллипс. Требуемое распределение точек порождается равномерными приращениями параметра θ.

Рассматривая производные x и y:

$$dy = bcos\theta d\theta, \quad dx = -asin\theta d\theta \tag{6.4.2}$$

получаем, что при θ, близком к 0 или π, т.е. у концов, $|dx| \approx 0$ и $|dy| \approx bd\theta$. Если θ близко к $\pi/2$ или $3\pi/2$, то $|dx| \approx ad\theta$ и $|dy| \approx 0$. Таким образом, около концов, где кривизна более высокая, точки располагаются чаще, а вдоль сторон, где кривизна меньше, - реже. Отношение приращений периметра концов к приращениям вдоль сторон приблизительно равно b/a. Отметим, что для окружности$(b = a)$ достигается оптимальное представление - равные приращения параметра или угла.

Если задано фиксированное количество точек на эллипсе, можно, пользуясь формулами суммы углов, получить эффективный алгоритм. Координаты любой точки на эллипсе:

$x_i = \mathrm{acos}(\theta_i + \Delta\theta), \quad y_i = \mathrm{bsin}(\theta_i + \Delta\theta),$

где $\Delta\theta = 2\pi/(n-1)$ - фиксированное приращение θ , n - количество точек на периметре, θ_i - значение параметра для точки x_i, y_i.

По формуле суммы углов

$$x_{i+1} = a(cos\theta_i cos\Delta\theta - sin\theta_i sin\Delta\theta), y_{i+1} = b(cos\theta_i sin\Delta\theta - sin\theta_i cos\Delta\theta)$$

Применяя уравнение (6.3.1) с $\theta = \theta_i$, получим уравнения:

$$x_{i+1} = x_i cos\Delta\theta - \left(\frac{a}{b}\right) y_i sin\Delta\theta,$$

$$y_{i+1} = \left(\frac{b}{a}\right) x_i sin\Delta\theta + y_i cos\Delta\theta. \tag{6.4.2}$$

Так как $\Delta\theta$, a и b - константы, полученный алгоритм содержит во внутреннем цикле только четыре умножения, одно сложение и вычитание и достаточно эффективен. Алгоритм дает многоугольник максимальной площади, вписанный в эллипс. Результат представлен на рис. 6.4.2.

Чтобы получить эллипс с центром не в начале координат и с главной осью, расположенной под углом к горизонтали, его поворачивают вокруг начала координат, а затем переносят.

Пример 2 Параметрическое построение эллипса

Построить эллипс с большой полуосью $a = 4$ и малой полуосью $b = 1$, под углом 30^0 к горизонтали, с центром в точке (2, 2).

Сначала построим эллипс с центром в начале координат с помощью 32 точек (*n=33*, так как первая и последняя точки совпадают). Рассмотреть достаточно только точки первого квадранта, т.е. диапазон параметра от 0 до $\pi/2$.

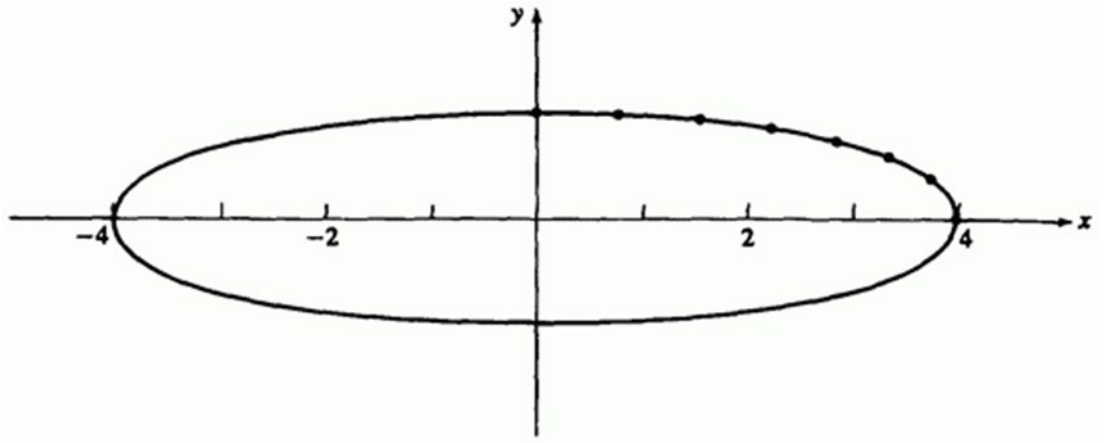


Рис.6.4.2. Параметрическое построение эллипса

Приращение параметра

$$\Delta\theta = \frac{2\pi}{(n-1)} = \frac{2\pi}{32} = \frac{\pi}{16}.$$

Пользуясь уравнением (6.4.1) с начальным значением $\theta = 0$, найдем x и y:

$$x_1 = a\cos\theta_1 = (4)\cos(0) = 4,$$

$$y_1 = b\sin\theta_1 = (1)\sin(0) = 0.$$

Вычислим величины $\frac{a}{b} = 4$, $\frac{b}{a} = 1/4$ и

$$\sin\Delta\theta = \sin\frac{\pi}{16} = 0.195,$$

$$\cos\Delta\theta = \cos\frac{\pi}{16} = 0.981.$$

Теперь по формуле (6.4.2) найдем вторую точку

$$x_2 = x_1\cos\Delta\theta - \left(\frac{a}{b}\right)y_1\sin\Delta\theta = (4)(0.981) - (4)(0)(0.195) = 3.92,$$

$$y_2 = \left(\frac{b}{a}\right)x_1\sin\Delta\theta + y_1\cos\Delta\theta = \left(\frac{1}{4}\right)(4)(0.195) + (0)(0.981) = 0.195.$$

Результаты для оставшихся точек первого квадранта представлены в таблице 6.4.1 и на рис. 6.4.2.

Нам требуется комбинированное двумерное преобразование поворота на 30^0 вокруг центра координат и переноса центра в точку (2, 2).

Характерно, что

$$[S] = [R][T] = \begin{bmatrix} cos\alpha & sin\alpha & 0 \\ -sin\alpha & cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ m & n & 1 \end{bmatrix}$$
$$= \begin{bmatrix} 0.866 & 0.5 & 0 \\ -0.5 & 0.866 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 0.866 & 0.5 & 0 \\ -0.5 & 0.866 & 0 \\ 2 & 2 & 1 \end{bmatrix}.$$

Применим это преобразование для точек (x_1, y_2) и(x_1, y_2):

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{bmatrix} [S] = \begin{bmatrix} 4 & 0 & 1 \\ 3.923 & 0.195 & 1 \end{bmatrix} \begin{bmatrix} 0.866 & 0.5 & 0 \\ -0.5 & 0.866 & 0 \\ 2 & 2 & 1 \end{bmatrix}$$
$$= \begin{bmatrix} 5.464 & 4 & 1 \\ 5.3 & 4.131 & 1 \end{bmatrix}.$$

Результаты приведены в табл. 6.4.2.

Таблица 6.4.1 Точки эллипса с центром в начале координат для первого квадранта

i	x_i	y_i
1	4.0	0
2	3.923	0.195
3	3.696	0.383
4	3.326	0.556
5	2.828	0.707
6	2.222	0.831
7	1.531	0.924
8	0.780	0.981
9	0	1.0

Таблица 6.4.2 Повернутый и перенесенный эллипс

i	x_i	y_i
1	5.464	4.0
2	5.3	4.131
3	5.009	4.179

4	4.603	4.144
5	4.096	4.027
6	3.509	3.831
7	3.864	3.565
8	2.185	3.240
9	1.5	2.866

6.5. Генерация файла для геометрического объекта «тор»

Рассмотрим пример, в котором координаты всех вершин вычисляются по ограниченному количеству данных. В качестве такого примера рассмотрим

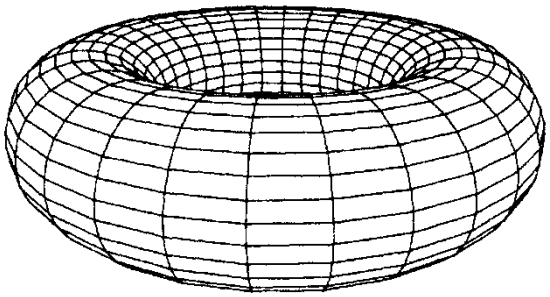


Рис.6.1.1.Тор

тор, изображенный на рис.6.1.1. Входные данные для этой программы состоят из трех чисел n, R и $r(R > r)$.

На рис.4.2 большая горизонтальная окружность определяет положение центров окружностей, образующих тор, радиус этой окружности равен $\boldsymbol{R}$. Выберем $\boldsymbol{n}$ равноудаленных точек на этой окружности в качестве центров малых вертикально расположенных окружностей радиуса $\boldsymbol{r}$. Параметрическое представление большой окружности описывается формулами

$$x = R\cos\alpha$$
$$y = R\sin\alpha$$
$$z = 0$$

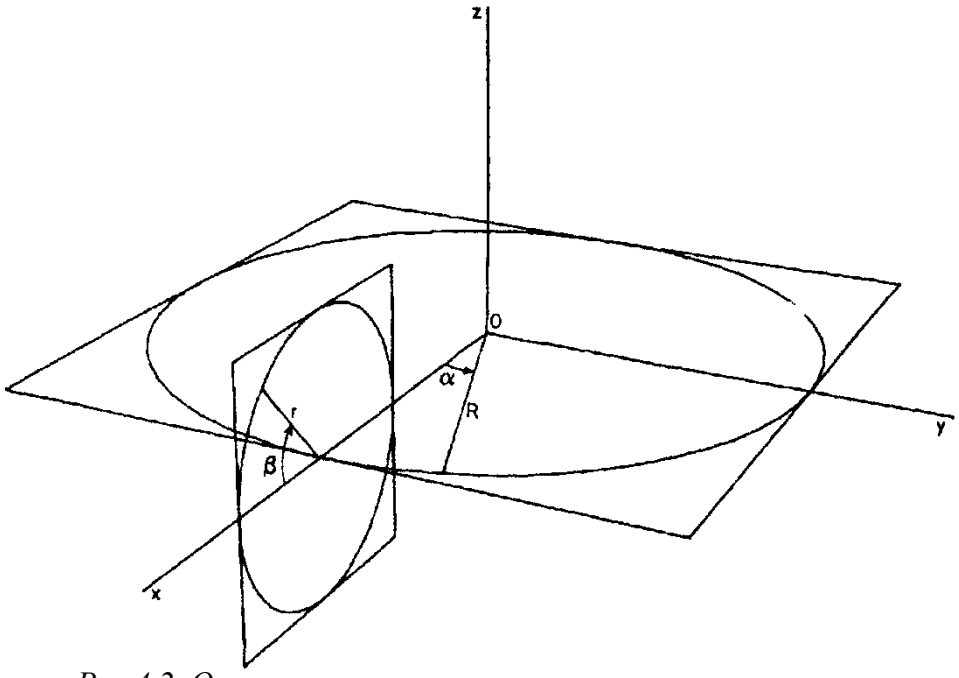


Рис.4.2. Основные окружности тора

Точка, соответствующая $\alpha = 0$, является центром малой окружности

$$x = R + r\,cos\,\beta$$
$$y = 0$$
$$z = r\,sin\,\beta$$

которая также показана на рис. 4.2. Остальные $n-1$ малые окружности формируются поворотом этой исходной окружности вокруг оси z на угол $\alpha = i\delta$, где $i = 1,...,n-1$ и $\delta = 2\pi / n$.

На малой окружности выберем $\boldsymbol{n}$ точек с номерами вершин $0,1,...,n-1$. Размещение точек на первой малой окружности определяется параметром $\beta = j\delta$, им приписываются номера вершин $j(\,j = 0,1,...,n-1)$. Следующие $\boldsymbol{n}$ вершин, пронумерованные $n, n+1,...,2n-1$, лежат на соседней малой окружности, соответствующей $i = 1$, и так далее. В общем, мы получим номера вершин $i \cdot n + j(i = 0,1,...,n-1, j = 0,1,...,n-1)$. Поворот на угол $\boldsymbol{\alpha = i\delta}$ относительно оси z записывается как

$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} cos\,\alpha & sin\,\alpha \\ -sin\,\alpha & cos\,\alpha \end{bmatrix}$$

В нашей ситуации исходная малая окружность лежит в плоскости $\boldsymbol{xOz}$, поэтому $\boldsymbol{y = 0}$, что сокращает это матричное произведение до

$$x' = x\cos\alpha$$
$$y' = y\sin\alpha$$

Этот же результат может быть получен непосредственно из рис. 4.2.

Следующая программа генерирует файл для вычерчивания тора.

```
/* TORUS (preprocessor for TOR generated) */
/* Препроцессор для генерации тора */
#include "stdio.h"
#include "math.h"
main()
{ FILE *fp;
int i, j, n;
float r, R, pi, alpha, beta, cosa, sina, x, x1, y1, z1, delta;
printf /* "Give number n (to draw an n x n torus)" */
("Задайте число n (для вычерчивания тора из n x n сегментов):\n");
scanf("%d", &n):
printf /* "Give large radius R and small radius r" */
    ("Задайте большой (R) и малый (г) радиусы тора\n");
scanf("%f %f",&R,&r);
fp=fopen("torus.dat", "w");
pi=4.0*atan(1.0); delta=2.0*pi/n;
fprintf(fp, "0.0 0.0 0.0\n"); /* central object point */
/* центральная точка объекта */
for (i=0;i<n; i++)
   { alpha=i*delta; cosa=cos(alpha); sina=sin(alpha);
       for (j=0; j<n; j++)
           {    beta=j*delta; x=R+r*cos(beta);          /*y=0 */
                x1=cosa*x; y1=sina*x; z1=r*sin(beta);    /* z1= z */
                fprintf(fp, "%d %e %e %e\n", i*n+j, x1, y1, z1);
```

```
                    /* %f not correctly implemented */
                    /* применение формата %f здесь не очень корректно */
            }
    }
fprintf(fp, "Faces – Грани:\n");
for (i-0; i<n;i++)
for(j-0; j<n; j++)
    { fprintf(fp, "%d %d %d %d#\n",
            i*n+j, (i+1)%n*n+j, (i+1)%n*n+(j+1)%n, i*n+(j+1)%n;
    }
fclose(fp);
}
```

7. ИНТЕРПОЛЯЦИЯ

Задача приближения функций возникает при обработке и отображении экспериментальных данных и в ходе моделирования и отображения геометрии сложных криволинейных объектов.

Простейшая, одномерная задача интерполяции, приводящая к приближению функций, заключается в следующем: *в дискретные моменты* $x_1, x_2, ..., x_n$ *наблюдаются значения функции* $y = f(x)$; *требуется восстановить её значение при других* x, *т.е. определить такую функцию* g, *чтобы* $f(x) \approx g(x; a_1, a_2, ..., a_n)$, *где* $a_1, a_2, ..., a_n$ *определяются из условия совпадения функции* $g(x)$ *и точках* $x_k, g(x) = f(x_k), k = 1,2,...,n$.

7.1 Кривые Безье

Кривы́е Безье́ или *Кривы́е Бернште́йна-Безье́* были разработаны в 60-х годах XX века независимо друг от друга *Пьером Безье* (*Pierre Bézier*) из автомобилестроительной компании «*Рено*» и *Полем де Кастельжо* (*Paul de Faget de Casteljau*) из компании «*Ситроен*», где применялись для проектирования кузовов автомобилей.

Несмотря на то, что открытие Полем де Кастельжо было сделано несколько ранее Пьера Безье (1959 г.), его исследования не публиковались и скрывались компанией как производственная тайна до конца 1960-х.

В основе кривая Безье является частным случаем многочленов Бернштейна, описанных *Сергеем Натановичем Бернштейном* в 1912 году.

Впервые кривые были представлены широкой публике в 1962 году французским инженером Пьером Безье, который, разработав их независимо от Поля де Кастельжо, использовал их для компьютерного проектирования автомобильных кузовов. Кривые были названы именем Безье, а именем де

Кастельжо назван разработанный им рекурсивный способ определения кривых (*алгоритм де Кастельжо*).

Впоследствии это открытие стало одним из важнейших инструментов систем автоматизированного проектирования и графических функций компьютерной графики.

Определение. Кривая Безье — параметрическая кривая, задаваемая выражением

$$B(t)=\sum_{i=0}^{n} P_i b_{i,n}(t), \quad 0 \le t \le 1$$

где P_i — функция компонент векторов опорных вершин, а $b_{i,n}(t)$ — базисные функции кривой Безье, называемые также полиномами Бернштейна.

$$b_i(t)=\binom{n}{i} t^i (1-t)^{n-i},$$

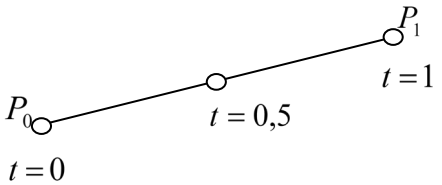
где $\binom{n}{i}=\frac{n!}{i!(n-i)!}$ — число сочетаний из n по i, где n — степень полинома, i — порядковый номер опорной вершины.

Виды кривых Безье

Линейные кривые.

При n = 1 кривая представляет собой отрезок прямой линии, опорные точки P_0 и P_1 определяют его начало и конец. Кривая задаётся уравнением:

$$B(t)=(1-t)P_0+tP_1 \quad t \in [0,1].$$

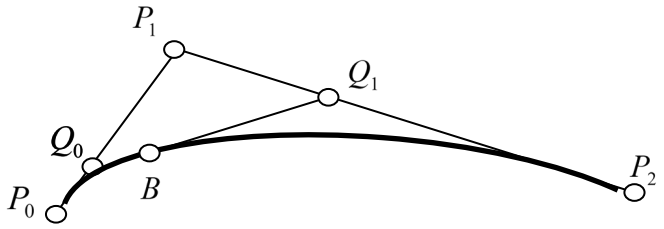


Квадратичные кривые.

Квадратичная кривая Безье (*n=2*) задаётся 3-мя опорными точками: P_0, P_1, P_2:

$$B(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2, \quad t \in [0,1].$$

Алгоритм Кастельджо.



- Точка Q_0 изменяется от P_0 до P_1 и описывает линейную кривую Безье;
- Точка Q_1 изменяется от P_1 до P_2 и также описывает линейную кривую Безье;
- Точка B изменяется от Q_0 до Q_1 и описывает квадратичную кривую Безье.

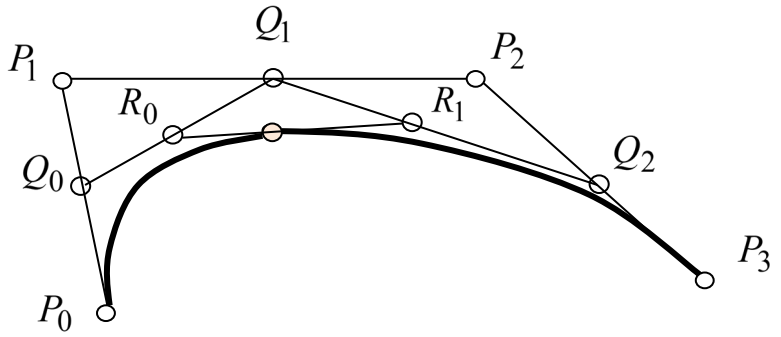
Кубические кривые.

В параметрической форме кубическая кривая Безье (n = 3) описывается следующим уравнением:

$$B(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3, \quad t \in [0,1].$$

Четыре опорные точки P_0, P_1, P_2, P_3, заданные в 2-х или 3-мерном пространстве определяют форму кривой.

Алгоритм Кастельджо.



Для построения кривых высших порядков соответственно требуется и больше промежуточных точек. Для кубической кривой это промежуточные точки Q_0, Q_1, Q_2, описывающие линейные кривые, а также точки R_0, R_1, которые описывают квадратичные кривые. Более простое уравнение имеет вид:

$$\frac{P_0Q_0}{P_0P_1} = \frac{P_1Q_1}{P_1P_2} = \frac{BQ_0}{Q_1Q_0}$$

Кривая берёт начало из точки P0 направляясь к P1 и заканчивается в точке P3 подходя к ней со стороны P2. То есть кривая не проходит через точки P1 и P2, они используются для указания её направления. Длина отрезка между P0 и P1 определяет, как скоро кривая повернёт к P3.

7.2. Рациональные кривые Безье

По заданному массиву вершин $P_0, \dots, P_1$ *(элементарная) рациональная кривая Безье степени m* определяется уравнением следующего вида:

$$R(t) = \frac{\sum_{\iota=0}^{m} w_\iota B_\iota^m(t) P_\iota}{\sum_{\iota=0}^{m} w_\iota B_\iota^m(t)},$$

где $B_i^m(t)$- многочлены Бернштейна.

неотрицательные числа w_i сумма которых положительна, называются *весами*. Если все веса w_i равны между собой, получается стандартная элементарная кривая Безье *m*-й степени.

Свойства рациональных кривых Безье

Элементарная рациональная кривая Безье, порождённая массивом **P**:

1+ - является гладкой кривой;

2+ - начинается в первой вершине P_0 массива **P,** $\boldsymbol{R(0) = P_0}$**,** касаясь отрезка P_0P_1 опорной ломаной

$$R(0) = m\frac{w_1}{w_0}(P_1 - P_0),$$

и заканчивается в последней точке P_m, $\boldsymbol{R(1) = P_m}$**,** касаясь отрезка $P_{m-1}P_m$ опорной ломанной,

$$R(1) = m\frac{w_{m-1}}{w_m}(P_m - P_{m-1});$$

3+ - лежит в выпуклой оболочке, порождённой массивом опорных вершин **P;**

4+ - симметрична – при перемене порядка вершин массива на противоположный,

$$P_0P_1, \dots, P_{m-1}P_m \to P_mP_{m-1}, \dots, P_1P_0,$$

не изменяет своей формы;

5+ - афинно-инвариантна;

6+ - «повторяет» контрольную ломанную (в частности, число точек пересечения рациональной кривой Безье с произвольной прямой не больше числа точек пересечения с этой прямой контрольной ломанной);

7+ - в случае, если опорные вершины $P_0, \dots, P_m$ лежат на одной прямой (коллинеарны), рациональная кривая Безье совпадает с отрезком P_0P_m;

8+ - в случае, если опорные вершины $P_0, \dots, P_m$ лежат в одной плоскости (компланарны), рациональная кривая Безье также лежит в этой плоскости;

9+ - Элементарная рациональная кривая Безье проективно-инвариантна;

10+ - поведение рациональной кривой Безье определяется не только массивом вершин, но и набором свободных параметров – весов w_i, *параметров формы;*

11- - степень функциональных коэффициентов напрямую связана с количеством вершин в массиве (на единицу больше) и растёт при его увеличении;

12- - при добавлении в массив хотя бы одной вершины возникает необходимость полного пересчёта параметрических уравнений элементарной кривой Безье;

13- - изменение хотябы одной вершины в массиве приводит заметному изменению всей кривой Безье.

Свойство рациональной кубической кривой Безье

Рассмотрим два набора вершин: $P_0^{(1)}, P_1^{(1)}, P_2^{(1)}, P_3^{(1)}$ и $P_0^{(2)}, P_1^{(2)}, P_2^{(2)}, P_3^{(2)}$.

При условии, что вершины $P_3^{(1)}$ и $P_0^{(2)}$ совпадают, рациональные кубические кривые $\gamma^{(1)}$ и $\gamma^{(2)}$, порождённые этими наборами, имеют общую точку.

Пусть $\gamma = \gamma^{(1)} \cup \gamma^{(2)}$ - полученная составная кривая.

При условии, что вершины

$$P_2^{(1)}, P_3^{(1)} = P_0^{(2)}, P_1^{(2)}$$

коллинеарны, подбором весов

$$w^{(1)} = \left(w_0^{(1)}, w_1^{(1)}, w_2^{(1)}, w_0^{(1)}\right), \quad w^{(2)} = \left(w_0^{(2)}, w_1^{(2)}, w_2^{(2)}, w_3^{(2)}\right)$$

можно добиться непрерывного изменения касательного вектора вдоль кривой γ. Достаточно положить

$$\frac{w_2^{(1)}}{w_3^{(1)}}\left|P_3^{(1)} - P_2^{(1)}\right| = \frac{w_1^{(2)}}{w_0^{(2)}}\left|P_1^{(2)} - P_0^{(2)}\right|$$

при условии, что вершины

$$P_1^{(1)}, P_2^{(2)}, P_3^{(1)} = P_0^{(2)}, P_1^{(2)}, P_2^{(2)}$$

компланарны (лежат в одной плоскости), т.е. векторы

$$\left(P_2^{(1)} - P_1^{(1)}\right) \times \left(P_3^{(1)} - P_2^{(1)}\right), \qquad \left(P_1^{(2)} - P_0^{(2)}\right) \times \left(P_2^{(2)} - P_1^{(2)}\right)$$

коллинеарны, подбором весов $w^{(1)}$ и $w^{(2)}$ можно добиться непрерывности вектора кривизны вдоль кривой γ. Их нужно взять так, чтобы выполнялось равенство

$$\frac{W^{(1)}}{\left|P_3^{(1)} - P_2^{(1)}\right|^3} = \frac{W^{(2)}}{\left|P_1^{(2)} - P_0^{(2)}\right|^3},$$

где

$$W^{(i)} = \frac{w_{2-i}^{(i)} w_{4-i}^{(i)}}{\left(w_{3-i}^{(i)}\right)^2}\left|\left(P_{3-i}^{(i)} - P_{2-i}^{(i)}\right) \times \left(P_{4-i}^{(i)} - P_{(3-i)}^{i}\right)\right|, i = 1,2.$$

Это свойство рациональных кубических кривых Безье позволяет поместить элементарную рациональную кубическую кривую Безье в разрыв между любыми двумя заданными (уже построенными) C^2-регулярными кривыми (в частности, элементарными рациональными кубическими кривыми Безье) так, что получаемая в результате составная кривая будет иметь непрерывный касательный вектор и непрерывный вектор кривизны.

Задача. По заданным набору из четырёх вершин P_0, P_1, P_2, P_3 и двум неотрицательным числам k_0 и k_3 найти веса w_0, w_1, w_2, w_3 так, чтобы значения кривизны рациональной кубической кривой Безье, порождаемой заданным массивом, в концах опорной ломанной P_0, P_1, P_2, P_3 совпадали с заданными числами (с k_0 в вершине P_0 и с k_3 в вершине P_3).

Решением задачи является набор $w = (w_0, w_1, w_2, w_3)$,

где

$$w_0 = 1, w_1 = \frac{4}{3}\left(\frac{c_0^2 c_3}{k_0^2 k_3}\right)^{\frac{1}{3}}, w_2 = \frac{4}{3}\left(\frac{c_0 c_3^2}{k_0 k_3^2}\right)^{\frac{1}{3}}, w_3 = 1,$$

$$c_0 = P\frac{|(P_1 - P_0) \times (P_2 - P_0)|}{2|P_1 - P_0|^3}, \quad c_3 = P\frac{|(P_2 - P_1) \times (P_3 - P_1)|}{2|P_3 - P_2|^3}$$

Замечание

Для планарного набора P_0, P_1, P_2, P_3 опорная ломанная которого принимает форму буквы S (рис. 7.2.1), числа k_0 и k_3 должны иметь разные знаки (кривизна кривой Безье меняет знак).

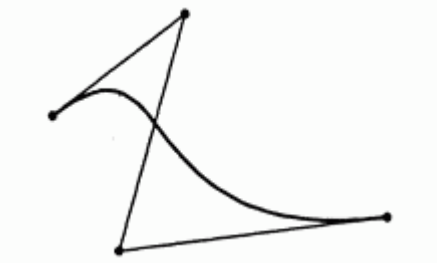


Рис.7.2.1. Кривая Безье с изменением знака кривизны

7.3. Интерполяция полиномами

Линейная интерполяция полиномами изображена на рисунке 7.2.1.а и в основе применяет линейный вид многочлена $g(x) = a_1 + a_2 x$. Чтобы прямая, описанная таким уравнением прошла через точки (x_1, y_1) и (x_2, y_2) достаточно выразить коэффициенты a_1 и a_2 через соответствующие уравнения системы:

$$\begin{cases} a_1 + a_2 x_1 = y_1 \\ a_1 + a_2 x_2 = y_2 \end{cases}.$$

Рисунок 7.2.1.б отображает квадратичный полином вида $g(x) = a_1 + a_2 x + a_3 x^2$ (1), при этом, коэффициенты определяются исходя из условия

$$\begin{cases} a_1 + a_2 x_1 + a_3 x_1^2 = y_1 \\ a_1 + a_2 x_2 + a_3 x_2^2 = y_2 \\ a_1 + a_2 x_3 + a_3 x_3^2 = y_3 \end{cases}, \quad (1)$$

где $(x_1, y_1), (x_2, y_2), (x_3, y_3)$- узловые точки, определяющие параболу участка кривой (Рис.7.2.1.б).

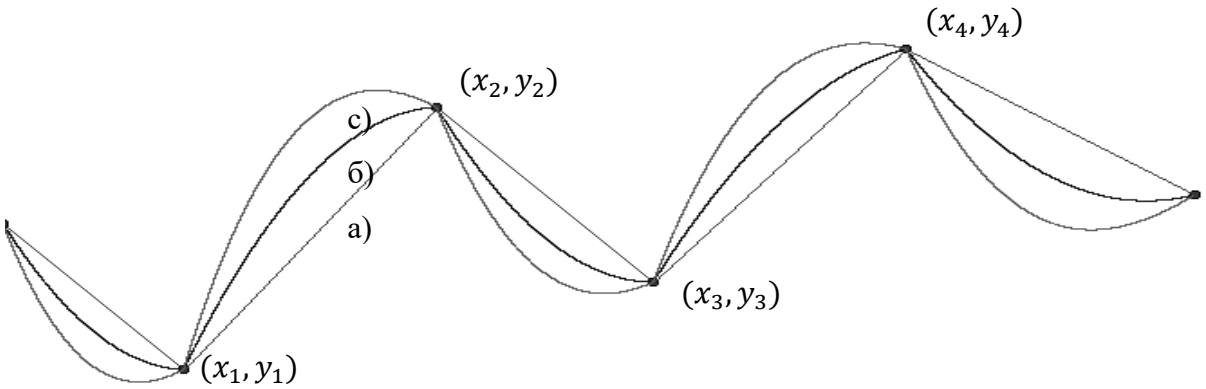


Рис.7.2.1. Интерполяция точечного каркаса функции полиномами:
а) первой степени; б) второй степени; с) третьей степени.

По аналогичным принципам задаётся кубический полином (рис.7.2.1.с), отображённый на этом же рисунке с большими значениями кривизны.

7.4. Гладкий кубический сплайн

Основной недостаток интерполяционных полиномов, приближающих функции, заключается в том, что поведение функции в окрестности какой-либо точки определяет поведение её в целом.

За счёт этого знак кривизны в узловой точке для соседних сегментов, как правило, меняется на противоположный (рис.7.2.1), а значит, узловая точка стремится к перегибу и исходная сетка не соответствует приближению зрительного восприятия функции через её каркас.

Аппаратом приближения, свободным от этого недостатка, являются сплайны.

Пусть две точки P_1 и P_2, показанные на рисунке 7.3.1, нужно соединить кривой S таким образом, чтобы она проходила через эти точки и гладко сопрягалась с соседними участками кривой. Соседние участки на рисунке показаны толстыми сплошными линиями, а желаемый вид кривой S – штриховой линией.

Очевидно, для решения задачи нужно наложить на кривую S четыре ограничения:

- кривая S проходит через точку P_1;
- кривая S проходит через точку P_2;
- сопряжение кривой S с соседним участком в точке P_1 является гладким;
- сопряжение кривой S с соседним участком в точке P_2 является гладким.

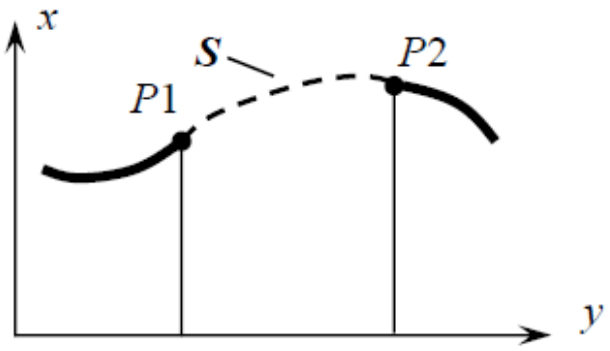


Рис.7.3.1. Интерполяция гладким кубическим сплайном

Для этого требуется иметь математическое описание кривой S в виде многочлена, имеющего не менее четырех коэффициентов. Самым простым из таких многочленов является многочлен третьей степени, который, в общем случае, имеет вид

$$y = ax^3 + bx^2 + cx + d, \quad (2)$$

включающий как раз четыре коэффициента формы *a, b, c* и *d*. Тогда наложение четырех упомянутых ограничений дает систему четырех уравнений с четырьмя неизвестными:

1) прохождение S через $P_1 - y_1 = ax_1^3 + bx_1^2 + cx_1 + d$;
2) прохождение S через $P_2 - y_2 = ax_2^3 + bx_2^2 + cx_2 + d$;
3) равенство первых производных соседних сплайнов в точке P_1-

$$y'_1 = 3ax_1^2 + 2bx_1 + c;$$

4) равенство первых производных соседних сплайнов в точке P_1-

$$y'_2 = 3ax_2^2 + 2bx_2 + c.$$

Теперь предположим, что кубическая парабола задана в параметрической форме (Рис.7.3.2)

$P(u) = \sum_{i=0}^{3} a_i u^i$, при $u = [0,1]$ или $P(u) = a_0 + a_1 u + a_2 u^2 + a_3 u^3$ (3)

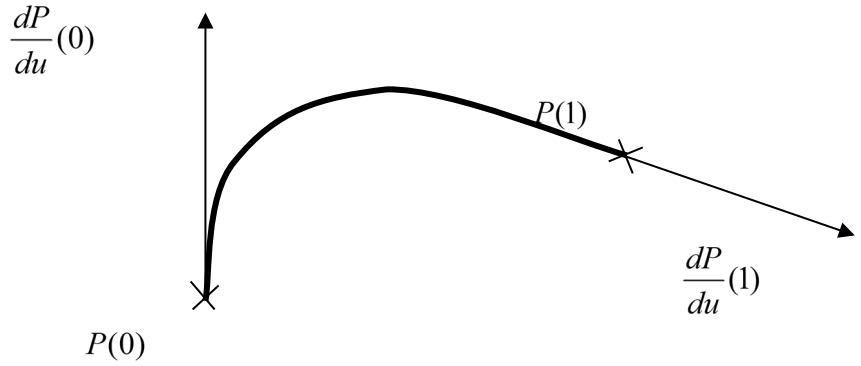


Рис.7.3.2 Определение кубического сплайна, заданного параметрически

проходит через две точки (рис.7.3.2), обозначенные соответственно $P(0)$ и $P(1)$, в которых известны значения производных $dP/du\,(0)$ и $dP/du\,(1)$. Это означает, что заданы четыре необходимых и достаточных условия для определения четырех коэффициентов в выражении (3).

Если определены четыре условия, то вычисление коэффициентов в выражении (3) выполняется довольно просто:

$$P(0) = a_0;$$

$$P(1) = \sum_{i=0}^{3} a_i = a_0 + a_1 + a_2 + a_3;$$

$$\frac{dP}{du}(0) = a_1;$$

$$\frac{dP}{du}(1) = \sum_{i=0}^{3} i a_i = a_1 + 2a_2 + 3a_3;$$

$$P(1) = P(0) + \frac{dP}{du}(0) + a_2 + a_3$$

и

$$\frac{dP}{du}(1) = \frac{dP}{du}(0) + 2a_2 + 3a_3.$$

Отсюда можно вывести

$$a_0 = P(0),$$

$$a_1 = \frac{dP}{du}(0),$$

$$a_2 = 3[P(1) - P(0)] - 2\frac{dP}{du}(0) - \frac{dP}{du}(1),$$

$$a_3 = 2[P(0) - P(1)] + \frac{dP}{du}(0) + \frac{dP}{du}(1).$$

Определив кубическую параболу (рис. 7.3.2) между точками $P(0)$ и $P(1)$, для нахождения следующей дуги кривой между точками $P'(0)$ и $P'(1)$ необходимо в точках $P(0)$ и $P(1)$ приравнять значения самой кривой и ее первых производных и задать значение вектора $\frac{dP'(1)}{du}$.

На рисунке 7.3.3 вектора для i-ой и $i+1$-ой точек определяются как $\frac{dP'(0)}{du} = \frac{y_{i+1} - y_{i-1}}{x_{i+1} - x_{i-1}}$, $\frac{dP'(1)}{du} = \frac{y_{i+2} - y_i}{x_{i+2} - x_i}$.

Таким образом, шаг за шагом определяется последовательность дуг кубической кривой, соединяющей точки $P, P^1, ..., P^n$ и имеющей непрерывные касательные в этих точках.

7.5. Сплайн-поверхность

Сплайновая поверхность, в отличие от кривой, должна проходить через четыре точки, являющиеся для нее угловыми. В общем случае, отсек сплайновой поверхности описывается бикубическими выражениями вида

$$\begin{aligned} x(u,v) = (C_{x00} + C_{x01}u + C_{x02}u^2 + C_{x03}u^3)v^0 + \\ +(C_{x10} + C_{x11}u + C_{x12}u^2 + C_{x13}u^3)v^1 + \\ +(C_{x20} + C_{x21}u + C_{x22}u^2 + C_{23}u^3)v^2 + \\ +(C_{x30} + C_{x31}u + C_{x32}u^2 + C_{33}u^3)v^3 = \\ = \sum_{i-0}^{3}\sum_{j=0}^{3} C_{xij}u^j v^i, \end{aligned}$$

$u = 0$ ...1 и $v = 0$ ... 1.

$C_{x00}, \dots, C_{x33}$ – коэффициенты формы, определяющие геометрические характеристики поверхности.

Аналогичный вид имеют выражения $y(u,v)$и $z(u,v)$. включая наборы коэффициентов $C_{y00}, \dots, C_{y33}$ и $C_{z00}, \dots, C_{z33}$ соответственно.

Смысл этих выражений следующий: аргументы u, v представляют собой координаты криволинейной координатной системы, расположенной на поверхности сплайна. В ней каждая точка поверхности задаётся парой числовых значений (рис. 7.4.1).

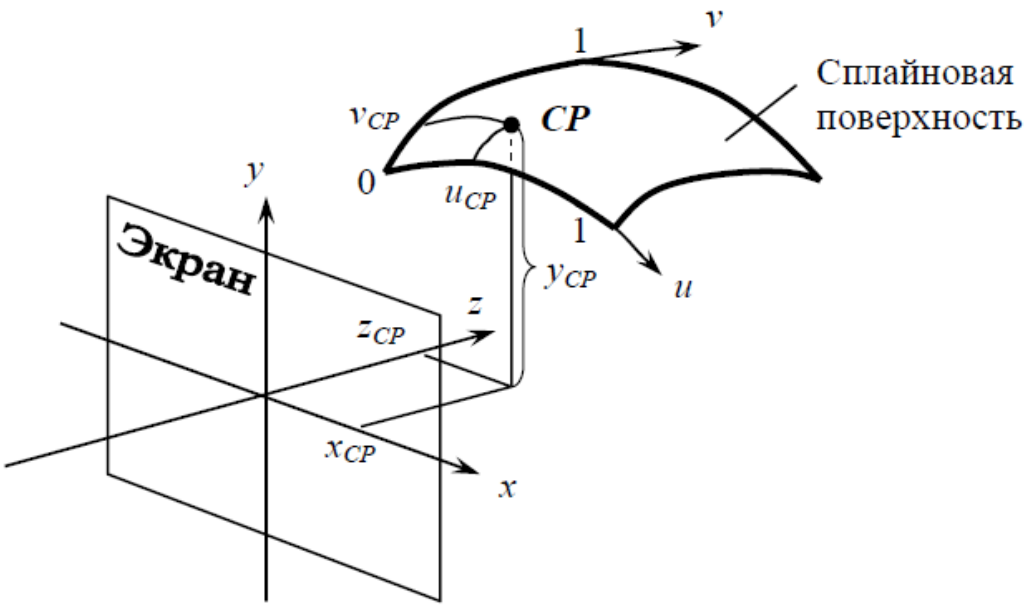


Рис.7.4.1. Соответствие между параметрами и декартовыми координатами.

Коэффициенты многочленов отыскиваются при наложении ограничений на форму отсека поверхности. В зависимости от выбора ограничений поверхность получает ту или иную форму описания.

В компьютерных системах геометрического моделирования в качестве ограничений обычно используется повторение сплайном формы некоторой многогранной опорной поверхности (характеристического многогранника), заданной шестнадцатью опорными точками. Поверхность должна проходить вблизи опорных точек или через некоторые из них, и изменение их координат должно приводить к изменению формы поверхности. Поэтому описание отсека поверхности представляют не в форме (3), а в другом виде, выражая коэффициенты многочленов через координаты опорных точек.

Описание в координатной форме имеет вид:

$$x(u,u) = \sum_{i=0}^{3}\sum_{j=0}^{3} f_i(v) f_j(u) P_{Xij},$$

$$y(u,u) = \sum_{i=0}^{3}\sum_{j=0}^{3} f_i(v) f_j(u) P_{Yij},$$

$$z(u,u) = \sum_{i=0}^{3}\sum_{j=0}^{3} f_i(v) f_j(u) P_{Zij},$$

где P_X, P_Y, P_Z– массивы X, Y, Z-координат точек;

$f_i(v), f_j(u)$– функциональные коэффициенты, имеющие свой вид для каждой разновидности сплайн-функции.

Для примера рассмотрим описание поверхности Безье:

$$f_0(u) = (1-u)^3 P_{x00} + 3(1-u)u^2 P_{x01} + 3u(1-u)^2 P_{x02} + u^3 P_{x03};$$

$$f_1(u) = (1-u)^3 P_{x10} + 3(1-u)u^2 P_{x11} + 3u(1-u)^2 P_{x12} + u^3 P_{x13};$$

$$f_2(u) = (1-u)^3 P_{x20} + 3(1-u)u^2 P_{x21} + 3u(1-u)^2 P_{x22} + u^3 P_{x23};$$

$$f_3(u) = (1-u)^3 P_{x30} + 3(1-u)u^2 P_{x31} + 3u(1-u)^2 P_{x32} + u^3 P_{x33};$$

$$x(u,v) = (1-v)^3 f_0(u) + 3(1-v)v^2 f_1(u) + 3v(1-v)^2 f_2(u) + v^3 f_3(u);$$

По аналогии описываются оставшиеся функции координат $y(u,v)$ и $z(u,v)$.

В компьютерной графике обычно применяется описание сплайна в матричной форме, которое выглядит следующим образом:

$$x(u,v) = U \cdot M \cdot P_X \cdot M^T \cdot V^T;$$

$$y(u,v) = U \cdot M \cdot P_Y \cdot M^T \cdot V^T;$$

$$z(u,v) = U \cdot M \cdot P_Z \cdot M^T \cdot V^T,$$

где U, V – векторы степеней параметров *u* и *v*:

$$U = [u^3 \quad u^2 \quad u \quad 1],$$

$$V = [v^3 \quad v^2 \quad v \quad 1],$$

$P_X, P_Y. P_Z$ – геометрические матрицы, содержащие координаты опорных точек, например, для нумерации координаты *X* опорных точек имеем

$$P_X = \begin{Bmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{Bmatrix}.$$

M – базисная матрица поверхности, которая содержит числовые коэффициенты, определяющие своеобразие поверхности.

Рассмотрим матричное представление на той же бикубической поверхности Безье. Ограничениями при построении этой поверхности является ее прохождение через угловые точки характеристического многогранника и заданные на его границах наклоны касательных в направлениях *u*, *v*. На рисунке 7.4.2 показана бикубическая поверхность Безье и ее характеристический многогранник, вершинами которого являются 16 опорных точек $P_{00}, \dots, P_{33}$.

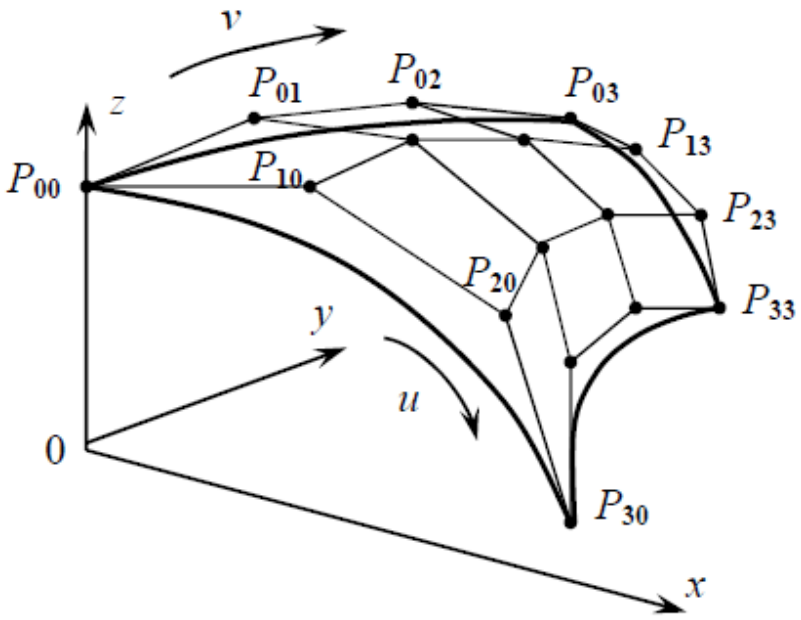


Рис.7.4.2. Представление бикубической поверхности Безье сеткой опорных точек.

Для поверхности Безье функциональные коэффициенты можно записать как

$$B(u) = \sum_{i}^{m} P_i b_{m,i}(u)$$

а значит

$$b_{m,i}(u) = C_m^i u^i (1-u)^{m-i} \text{ и } \; b_{n,j}(v) = C_n^j v^j (1-v)^{n-j},$$

где C_m^i, C_n^j – биноминальные коэффициенты.

$$C_m^i = \binom{m}{i} = \frac{m!}{i!\,(m-i)!}$$

при $m = 3$

$$\binom{3}{i} = \frac{6}{i!\,(3-i)!},$$

$$b_{3,0}(u) = (1)u^0(1-u)^3 = (1-u)^3$$

$$b_{3,1}(u) = 3u^1(1-u)^2$$

$$b_{3,2}(u) = 3u^2(1-u)^1$$

$$b_{3,3}(u) = u^3.$$

В матричной форме получаем для четырёх точек

$$B(u) = [(1-u)^3 \quad 3u(1-u)^2 \quad 3u^2(1-u) \quad u^3] \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix},$$

перегруппируя коэффициенты получим

$$B(u) = [U][M][P] = [u^3 \quad u^2 \quad u \quad 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

Аналогично, кривая Безье четвертого порядка *n=4*

$$B(u) = [U][M][P] = [u^4 \quad u^3 \quad u^2 \quad u \quad 1] \begin{bmatrix} -1 & -4 & 6 & -4 & 1 \\ -4 & 12 & -12 & 4 & 0 \\ 6 & -12 & 6 & 0 & 0 \\ -4 & 4 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}.$$

7.5. B-сплайновые кривые

Кривая, определяемая вершинами многоугольника, зависит от способов интерполяции или аппроксимации, устанавливающих связь кривой и многоугольника. Основой является выбор базисных функций. Базис Бернштейна описывает кривые Безье, но обладает двумя ограничениями гибкости кривых. Во-первых, количество вершин многоугольника жестко задает порядок многочлена и единственный способ понизить степень кривой

- это сократить количество вершин, а повысить степень кривой - увеличить их число.

Второе ограничение следует из глобальной природы базиса Бернштейна. Это означает, что величина аппроксимирующих функций $b_{n,i}(t)$ ненулевая для всех значений параметра на кривой, поэтому изменение какой-либо одной вершины оказывает влияние на всю кривую. Локальные воздействия на кривую невозможны.

Существует *неглобальный* базис, называемый *базисом B-сплайна*, включающий *базис Бернштейна* как частный случай. *B-сплайны* неглобальны, так как с каждой вершиной B_i связана своя базисная функция. Поэтому влияние каждой вершины на кривую проявляется только при тех значениях параметра, где соответствующая базисная функция не равна нулю. Базис *B-сплайна* также позволяет менять порядок базисных функций и, следовательно, всей кривой без изменения количества вершин. Рекурсивное определение для численного решения было выведено независимо Коксом и де Буром.

Пусть $B(t)$ определяет кривую как функцию от параметра *t*, тогда *B-сплайн* имеет вид

$$B(t) = \sum_{i=1}^{n+1} P_i N_{i,k}(t), \;\; t_{min} \le t \le t_{max}, \;\; 2 \le k \le n+1,$$

где $P_i \;\; - n+1$ вершина многоугольника, а $N_{i,k}(t)$ – нормализованные функции базиза B-сплайна.

Для *i*-й нормализованной функции базиса порядка *k* (степени *k-1*) функции базиса $N_{i,k}(t)$ определяются рекурсивными формулами Кокса-де Бура:

$$N_{i,k}(t) = \begin{cases} 1 & \text{если } x_i \le t < x_{i+1} \\ 0 & \text{иначе} \end{cases}$$

и

$$N_{i,k}(t) = \frac{(t - x_i)N_{i,k-1}(t)}{x_{i+k-1} - x_i} + \frac{(x_{i+1} - t)N_{i+1,k-1}(t)}{x_{i+k} - x_{i+1}}$$

Величины x_i – это элементы узлового вектора, удовлетворяющего отношению $x_i \le x_{i+1}$. Параметр *t* изменяется на промежутке $[t_{min}, t_{\max})$ вдоль кривой $B(t)$. При этом считаем, что 0=0/0.

7.5.1. Параметрическое уравнение элементарной кубической B-сплайновой кривой

По заданному массиву

$$P_0, P_1, P_2, P_3$$

(элементарная) кубическая B-сплайновая кривая определяется при помощи векторного уравнения, имеющего следующий вид:

$$R(t) = \frac{(1-t)^3}{6}P_0 + \frac{3t^3 - 6t^2 + 4}{6}P_1 + \frac{-3t^3 + 3t^2 + 3t + 1}{6}P_2 + \frac{t^3}{6}P_3,$$

$$0 \le t \le 1.$$

Матричная запись параметрических уравнений, описывающих элементарную кубическую *B*-сплайновую кривую,

$$R(t) = PMT, \quad 0 \le t \le 1,$$

где

$$R(t) = \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix}, \quad M = \frac{1}{6}\begin{pmatrix} 1 & -3 & 3 & -1 \\ 4 & 0 & -6 & 3 \\ 1 & 3 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad T = \begin{pmatrix} t^0 \\ t^1 \\ t^2 \\ t^3 \end{pmatrix},$$

$$P = (P_0 \quad P_1 \quad P_2 \quad P_3) = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ y_0 & y_1 & y_2 & y_3 \\ z_0 & z_1 & z_2 & z_3 \end{pmatrix}.$$

Матрица M назвыается базисной матрицей B-сплайновой кривой.

Свойства элементарных кубических B-сплайновых кривых

Свойства функциональных весовых множителей

$$n_0(t) = \frac{(1-t)^3}{6}, \qquad n_1(t) = \frac{3t^3 - 6t^2 + 4}{6},$$

$$n_2(t) = \frac{-3t^3 + 3t^2 + 3t + 1}{6}, n_3(t) = \frac{t^3}{6}$$

оказывают существенное влияние на поведение элементарной кубической B-сплайновой кривой. Вот некоторые из них.

Функциональные коэффициенты $n_i(t)$

1$^+$ неотрицательны,

2$^+$ в сумме составляют единицу,

3$^+$ не зависят от точек массива P_0, P_1, P_2, P_3 (универсальны).

Элементарная кубическая B-сплайновая кривая

1$^+$ лежит в выпуклой оболочке, полождённой вершинами P_0, P_1, P_2, P_3 опорной ломанной, и, как правило, не проходит ни через одну из них;

2$^+$ касательная в концевой точке

$$\frac{1}{6}(P_0 + 4P_1 + P_2)$$

параллельна отрезку P_0P_2, а в концевой точке

$$\frac{1}{6}(P_1 + 4P_2 + P_3)$$

- отрезку P_1P_3 (рис.7.6.1.1)

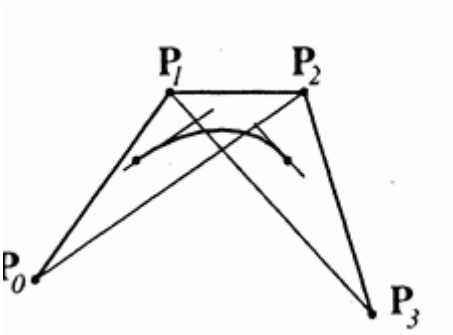


Рис.7.5.1.1.

Рассмотрим ещё один способ построения гладких кривых по форме *B-сплайна*. Будем использовать параметрическое представление кривых. Любая точка части кривой между двумя заданными последовательными точками P и Q будет иметь координаты $x(t)$ и $y(t)$, где t увеличивается от 0 до1, если отслеживается часть кривой от точки P до точки Q. Можно считать, что t - это время.

Если имеются заданные точки

$P_0(x_0, y_0), P_1(x_1, y_1) ... P_n(x_n, y_n)$

то часть кривой *B-сплайна* между двумя последовательными точками P_i и P_{i+1} получается путем вычисления функций $x(t)$ и $y(t)$ для изменения t от 0 до 1

$$x(t) = \{(a_3 t + a_2)t + a_1\}t + a_0$$

Эти уравнения содержат следующие коэффициенты:

$$a_3 = (-x_{i-1} + 3x_i - 3x_{i+1} + x_{i+2})/6$$

$$a_2 = (x_{i-1} - 2x_i + x_{i+1})/2$$

$$a_1 = (-x_{i-1} + x_{i+1})/2$$

$$a_0 = (x_{i-1} + 4x_i + x_{i+1})/6$$

а коэффициенты b_3, b_2, b_1, b_0 вычисляются по значениям $y_{i-1}, y_i, y_{i+1}, y_{i+2}$ аналогичным образом. Вышеприведенные формулы пригодны для эффективных вычислений. Вычисление значений $x(t)$ производится быстрее по правилу Горнера, чем по обычному полиномиальному выражению. Коэффициенты a_3, a_2, a_1, a_0 вычисляются только однажды для каждого сегмента кривой, что очень важно, поскольку на каждом сегменте кривой может вычисляться большое число промежуточных точек $x(t)$ и $y(t)$.

Для получения некоторого представления о свойствах кривой в точках стыковки двух сегментов рассмотрим функцию $x(t)$ и ее первую и вторую производные для значений $t = 0$ и $t = 1$ (функция $y(t)$ будет обладать аналогичными свойствами)

$$x(0) = a_0 = (x_{i-1} + 4x_i + x_{i+1})/6$$

$$x(1) = a_3 + a_2 + a_1 + a_0$$

Используя уравнение коэффициентов, после упрощения получаем

$$x(1) = (x_i + 4x_{i+1} + x_{i+2})/6$$

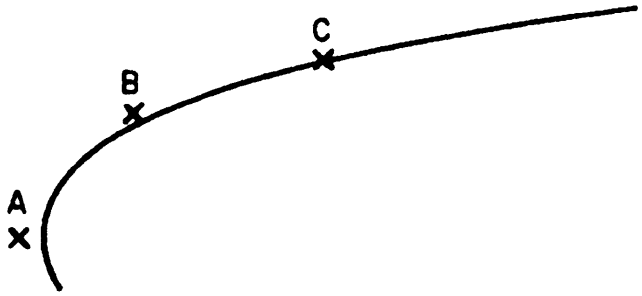
Можно видеть, что значение $x(0)$ не равно в точности x-координате x_i точки P_i: оно зависит от позиций точек P_{i-1} и P_{i+1}. Из рисунка 5.1.7 видно, что

точка B является точкой сегмента AB, но одновременно и начальной точкой сегмента BC.

Для первого сегмента имеем

$$A = P_i, B = P_{i+1}, C = P_{i+2}$$
$$x_B* = x(0) = (x_A + 4x_B + x_C)/6$$

Рис.5.1.7. Три последовательные точки



где через x_B* обозначено вычисленное значение координаты x для точки B. Рассматривая эту же точку как принадлежащую сегменту BC, получим

$$A = P_{i-1}, B = P_i, C = P_{i+1}$$
$$x_B* = x(0) = (x_A + 4x_B + x_C)/6$$

Отсюда видно, что оба способа вычисления значения x дают одинаковый результат, что означает непрерывность функции $x(t)$ в точке B. Продифференцировав $x(t)$ дважды, найдем производные $x'(t)$ и $x''(t)$. Подставляя в них значения $t = 0$ и $t = 1$, как это было сделано для $x(t)$, можно будет убедиться, что производные непрерывны в точке B. Поскольку функция $y(t)$ и ее первые

две производные тоже непрерывны, то становится ясно, что кривая B-сплайна очень гладкая.

Для расчета любого сегмента кривой между точками P_i. и P_{i+1} используются также точки P_{i-1} и P_{i+2}. Из этого следует, что первый сегмент кривой будет располагаться между точками P_1 и P_2, а последний — между точками P_{n-2} и P_{n-1}. Так что начальная и конечная точки всей кривой будут

располагаться вблизи P_1 и P_{n-1} но не вблизи P_0 и P_n. Приведенная ниже программа на C++, приведённая в работе [2] считывает числа

n

$x_0 \quad y_0$

$x_1 \quad y_1$

...

$x_n \quad y_n$

из файла CURV.DAT. При выводе каждая из $n+1$ точек обозначается маркером в виде креста. Затем вычерчивается кривая B- сплайна.

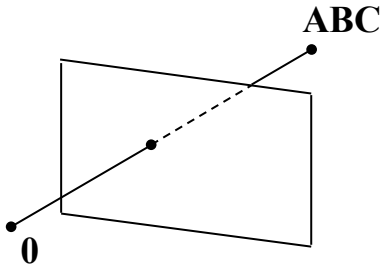
```
/* CURVFIT: Сглаживание кривой с применением B-сплайна */
/* Curve fitting using B splines */
#include "stdio.h"
#define MAX 100
#define N 30
main()
{ float x[MAX], y[MAX], eps=0.04, X, Y, t, xA, xB, xC, xD,
yA, yB, yC, yD, a0, a1. a2, a3, bO, b1, b2, b3;
int n, i, j, first;
FILE *fp;
Fp=fopen(“curv.dat", "r");
if (fp==NULL) { printf /* "There is no file curv.dat" */
                    ("Нет файла curv.dat\n”); exit(1); }
fscanf(fp, "%d", &n):
for(i=0;i<=n;i++)
if (fscanf(fp,"%f  %f", x+i, y+i)<=0)
{ printf                    /* "Reading beyond the end of file curv.dat" */
("Считывание выходит за пределы файла curv.dat\n”); exit(1);
}
initgr();
                    /* Mark the given points: */
```

```
                    /* Заданные точки отмечаются маркером: */
      for(i=0;i<=n;i++)
      {X=x[i];Y=y[i];
      move(X-eps, Y-eps); draw(X+eps, Y+eps);
      move(X+eps, Y-eps); draw(X-eps, Y+eps);
      }
      first=1;
for(i=1;i<n-1;i++)
{     xA=x[i-1]; xB=x{i]; xC=x[i+1]; xD=x[i+2];
       yA=y[i-1]; yB=y[i]; yC=y[i+1]; yD=y[i+2];
      a3=(-xA+3*(xB-xC)+xD)/6.0; b3=(-yA+3*(yB-yC)+yD)/6.0;
      a2=(xA-2*xB+xC)/2.0; b2=(yA-2*yB+yC)/2.0;
      a1=(xC-xA)/2.0; b1=(yC-yA)/2.0;
      a0=(xA+4*xB+xC)/6.0; b0=(yA+4*yB+yC)/6.0;
      for(j=0;j<=N;j++)
      {
      t=(float)j/(float)N;
      X=((a3*t+a2)*t+a1)*t+a0;
      Y=((b3*t+b2)*t+b1)*t+b0;
      it (first) { first=0; move(X, Y);} else draw(X, Y);
      }
}
 endgr();
fclose(fp);
}
```

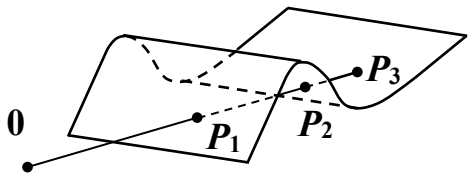
8. УДАЛЕНИЕ НЕВИДИМЫХ ЛИНИЙ И ПОВЕРХНОСТЕЙ.

8.1. Постановка проблемы.

Формализация приводит к следующей постановке задачи. Выделим в пространстве R^3 некоторую плоскость R^2. Пространство R^3 в дальнейшем будем называть объектным пространством, а плоскость R^2 - картинной плоскостью. Пусть в объектном пространстве находится некоторая поверхность S, расположенная по одну сторону от картинной плоскости. Будем считать, что наблюдатель расположен по другую сторону от картинной плоскости в центре проектирования (в случае ортогональной проекции центр проектирования бесконечно удален).

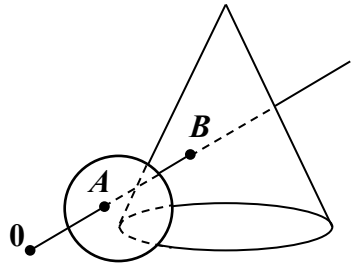


Будем говорить, что точка p_1 поверхности S загораживает точку p_2 этой поверхности, если проекции этих точек на картинную плоскость совпадают и при этом точка p_1 оказывается расположенной между p_2 и общей проекцией на картинную плоскость. *Определение.* Точка p поверхности называется видимой, если эта точка не загорожена никакой другой точкой поверхности.



Рассмотрим некоторые подходы к решению этой задачи, которую часто также называют задачей загораживания.

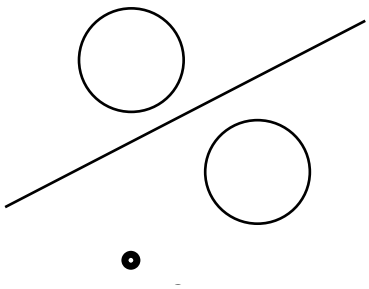
Эвристические соображения, которые могут служить отправной точкой для решения задачи загораживания:



1. Установка приоритета, в соответствии с которым нужно производить обработку данных для получения целостного изображения.

а). Пусть сцена разбита на два фрагмента – ***A*** и ***B***. Если самые дальние точки фрагмента ***A*** лежат ближе к наблюдателю, чем самые ближние точки фрагмента ***B*** , то части ***B*** не могут загородить фрагмента ***A***. Поэтому, если мы сначала построим изображение фрагмента ***B***, а затем – изображение фрагмента ***A***, то в результате правильно получим изображение всей сцены.

б). Пусть в объектном пространстве удалось найти плоскость, которая разделяет все пространство на такие два полупространства, в одном из которых находится фрагмент ***A*** и наблюдатель, а в другом – фрагмент ***B***. При этих условиях фрагмент ***B*** не может загораживать фрагмент ***A***. Остается лишь выявить, в каком полупространстве находится наблюдатель.



Если проекции фрагментов ***A*** и ***B*** не пересекаются, то обработка каждого из них может производиться независимо.

2. Установка факта независимости фрагментов.

Общая задача о пересечении проекций произвольных объектов весьма сложна, однако использование простых достаточных условий непересечения часто оказывается полезным. Например, описав вокруг проекций прямоугольники со сторонами, параллельными осям координат, и убедившись, что они не пересекаются (это сводится к проверке нескольких неравенств), мы можем быть уверены, что фигуры также не пересекаются.

Известные методы решения задачи загораживания различаются между собой по четырем основным характеристикам:

выбору структуры данных для представления поверхности;

пространству, в котором происходит анализ видимости;

способу визуализации поверхности;

использованию специфических геометрических свойств изображаемых объектов.

Важной характеристикой метода является "асимптотическое" время работы соответствующего алгоритма в зависимости от разрешения изображаемого

объекта и разрешения картинной плоскости. Это время может быть связано со свойствами изображаемой поверхности, внутренней структурой используемого алгоритма, а также другими факторами.

Представление поверхности может быть:

аналитическим – поверхность представлена неявно при помощи аналитического выражения; такое представление обычно используется для задания простых объектов – сфер, конусов, цилиндров и т.п.;

полиэдральным – поверхность представлена совокупностью многоугольных граней;

параметрическим – в виде набора частей, каждая из которых представляет собой параметрически заданную поверхность; нередко при помощи триангуляции такие поверхности заменяют их представлением в виде многогранника.

Под решением полиэдральной поверхности понимается количество ее граней.

Под разрешением картинной плоскости понимается количество точек растра (пикселов) на экране монитора.

Ребра многогранника и линии сетки параметрического представления называют каркасными линиями, а соответствующее изображение – каркасным. Если же поверхность изображается с использованием полутоновой закраски ее элементов, то такое изображение называют полутоновым, а сам процесс закраски – полутоновым заполнением (граней).

По типу пространства, в котором происходит анализ видимости, алгоритмы делятся на три группы:

Объектные – анализирующие взаимное расположение частей поверхности в объектном пространстве (временные характеристики таких алгоритмов обычно обладают квадратичной зависимостью от числа объектов сцены и их разрешения);

Картинные – определяющие видимость каждого элемента картинной плоскости (пиксела) в плоскости изображения (временные характеристики

таких алгоритмов оцениваются как линейные функции от произведения числа объектов на число точек растра);

Смешанные – использующие для анализа как первый, так и второй подходы.

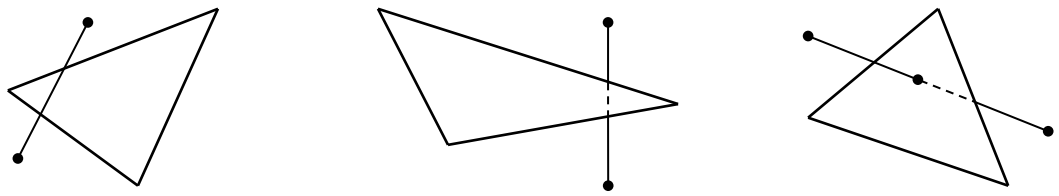
По способу визуализации алгоритмы загораживания разделяются на две группы, в одну из которых входят алгоритмы, ориентированные на получение ***каркасного***, а в другую – ***полутонового изображения***.

Под глубиной элемента поверхности понимается расстояние от этого элемента до картинной плоскости.

8.2. Некоторые подходы к решению задач загораживания.

8.2.1. Методы переборного типа. («грубой силы»).

В алгоритмах переборного типа, как правило, рассматривается каждое ребро, принадлежащее каждому из объектов сцены, и анализируется его взаимное расположение со всеми гранями каждого из объектов, составляющих сцену. При этом возможны следующие ситуации:



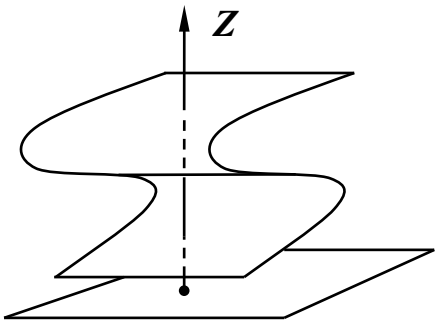
Временная сложность алгоритмов такого типа пропорциональна произведению количества ребер и количества граней в сцене. Никакие дополнительные свойства изображаемых объектов не учитываются.

8.2.2. Метод *Z*-буфера.

Другим методом «грубой силы» является метод ***Z***-буфера, который весьма удобен для аппаратной реализации ввиду простоты алгоритма и

используемого в нем набора операций. Временные характеристики этого метода линейно зависят от количества точек растра и «глубинной сложности сцены», т.е. усредненного числа граней, взаимно закрывающих друг друга.

Для реализации метода используются две области памяти: буфер глубины (***Z***-буфер) и буфер кадра (хранящий информацию о состоянии пикселов экрана компьютера). *Буфер глубины* используется для хранения координаты ***Z*** (глубины) каждого видимого на данной стадии анализа изображения пиксела картинной плоскости. В буфере кадра запоминаются атрибуты (интенсивность и цвет) соответствующего пикселя.



Формально описание метода таково. Предположим, что сцена представлена в виде объединения многоугольников (возможно, пересекающихся). Построим ортогональную проекцию сцены на картинную плоскость $z = 0$.

Предполагается следующая последовательность шагов:

1. Инициализировать буфер кадра фоновыми значениями интенсивности или цвета.
2. Инициализировать буфер глубины значениями глубины фона.
3. Для каждой грани сцены последовательно:

(4). Преобразовать проекцию границ в растровуюформу.

(5). Для каждого пиксела проекции вычислить его глубину $z = z(x, y)$;

(6). Сравнить значение $z(x, y)$ с соответствующим значением буфера глубины $Z(x, y)$.

(7). Если $z(x, y) < Z(x, y)$, то:

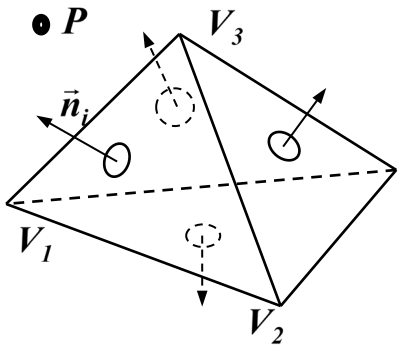
(8). Записать атрибуты этого пиксела в буфер кадра;

(9). Записать значение $z(x, y)$ в соответствующую позицию буфера глубины $Z(x, y)$.

(10). Иначе – никаких действий не производить.

8.3. Удаление нелицевых граней многогранника.

V1



Формальное описание метода.

Пусть $F_1, F_2, ..., F_N$ - грани многогранника. Рассмотрим одну из граней F_i.

Обозначим вершины, инцидентные грани F_i, через $V_1, V_2, ..., V_K$. Найдем вектор нормали к грани F_i, вычислив векторное произведение любых двух смежных ребер этой грани. Имеем:

$$n_i = [V_1V_2, V_2V_3].$$

Представим векторами выбранные ребра:

$$V_1V_2 = \vec{v} = (v_1, v_2, v_3)$$
$$V_2V_3 = \vec{v}' = (v_1', v_2', v_3')$$
$$\vec{n}_i = \vec{v} \times \vec{v}' = ((v_2v_3' - v_3v_2'), (v_3v_1' - v_1v_3'), (v_1v_2' - v_2v_1'))$$

Пусть $n_i = (A_i, B_i, C_i)$, тогда опорная функция грани имеет вид $L_i(p) = A_ix + B_iy + C_iz + D_i$ где $p = p(x, y, z)$ - центр проецирования.

Определим коэффициент D_i для плоскости, несущей грань F_i. Плоскость, определяемая вектором нормали и проходящая через одну из точек грани F_i (например V_1) описывается уравнением:

$$A_i(x - x_{V_1}) + B_i(y - y_{V_1}) + C(z - z_{V_1}) = 0$$
$$A_ix + B_iy + C_iz - (A_ix_{V_1} + B_iy_{V_1} + C_iz_{V_1}) = 0$$

отсюда можно записать скалярное произведение

$$D_i = -(n_i, V_1).$$

Тогда уравнение плоскости, несущей грань F_i можно записать как

$$L_i(x, y, z) = 0.$$

В случае, если многогранник является выпуклым, коэффициенты A_i, B_i и C_i легко выбрать так, чтобы вектор $n_i = (A_i, B_i, C_i)$ был вектором внешней нормали. Для этого определим какую-либо внутреннюю точку многогранника W, например, барицентр:

$W = (P_1 + P_2 + ... + P_m)/m$,

где P_1, P_2, … , P_m - множество всех вершин многогранника.

Положим $k_i = -sign(L_i(W))$, и далее:

$A_i = k_i A_i$, $B_i = k_i B_i$, $C_i = k_i C_i$, $D_i = k_i D_i$

Положительное R_+ и отрицательное R_- по отношению к грани F_i полупространства определяются соответственно неравенствами

$L_i(p) > 0$ и $L_i(p) < 0$, где p - точка центральной проекции P.

Сформулируем условия, определяющие, является ли грань лицевой.

В случае центрального проектирования грань F_i является лицевой, если $L_i(p) > 0$, и *нелицевой*, если $L_i(p) < 0$.

В случае ортогонального проектирования грань F_i - лицевая, если $(n_i, l) > 0$, и *нелицевая*, если $(n_i, l) < 0$, где l = вектор проектирования.

Для невыпуклых сложных фигур определение направления внешней нормали можно получить за счёт организации единого обхода точек каждой грани по часовой стрелке, рассматривая объект «снаружи». Процесс проецирования граней на двухмерное пространство приводит к противоположному изменению обхода точек для обратных граней. При этом реагирует знак детерминанта

$$D_i = \begin{vmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x_C & y_C & 1 \end{vmatrix} \quad \begin{array}{l} D > 0 - \textit{обход против часовой стрелки} \\ D < 0 - \textit{обход по часовой стрелке} \\ D = 0 - \textit{точки лежат на одной прямой} \end{array}$$

При этом усложняется описание объекта и требуется предварительная процедура проецирования каждой грани.

9. ПРОСТЕЙШИЕ МЕТОДЫ РЕНДЕРИНГА ПОЛИГОНАЛЬНЫХ МОДЕЛЕЙ

В большинстве случаев модели задаются набором плоских выпуклых граней. Поэтому при построении изображения естественно воспользоваться этой простой моделью. Существует три простейших метода рендеринга полигональных моделей, дающих достаточно приемлемые результаты, - метод плоского (постоянного закрашивания), метод Гуро и метод Фонга.

9.1. **Метод постоянного закрашивания**

Это самый простой метод из всех трёх. Он заключается в том, что на грани берётся произвольная точка и определяется её освещённость, которая принимается за освещённость всей грани.

Свет точечного источника отражается от идеального рассеивателя по закону косинусов Ламберта: интенсивность отражённого света пропорциональна косинусу угла между направлением света и нормалью к поверхности (рис.9.1.1), т.е.

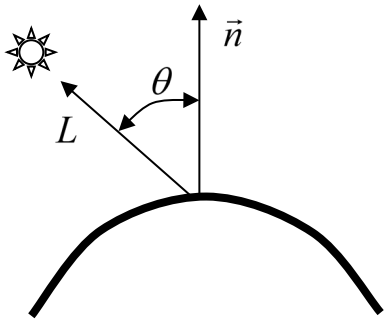


Рис.9.1.1.Дифузное отражение

$$I = I_l k_d \cos\theta \quad 0 \le \theta \le \pi / 2$$

где I - интенсивность отражённого света, I_l- интенсивность точечного источника, k_d- коэффициент диффузного отражения $(0 \le k_d \le 1)$, θ - угол между направлением света и нормалью к поверхности. Если $\theta > \pi / 2$, то источник света расположен за объектом. Коэффициент диффузного отражения k_d зависит от материала и длины волны света, но в простых моделях освещения обычно считается постоянным.

На объекты реальных сцен падает ещё и рассеянный свет, отражённый от окружающей обстановки. Поскольку для расчёта таких источников требуются большие вычислительные затраты, в компьютерной графике они заменяются на коэффициент рассеяния – константу, которая входит в формулу в линейной комбинации с членом Ламберта:

$I = I_a k_a + I_l k_d \cos\theta \quad 0 \le \theta \le \pi / 2$

где I_a - интенсивность рассеянного света, k_a - коэффициент диффузного отражения рассеянного света $(0 \le k_a \le 1)$.

Пусть даны два объекта, одинаково ориентированные относительно источника, но расположенные на разном расстоянии от него. Если найти их интенсивность по данной формуле, то она окажется одинаковой. Это означает, что когда предметы перекрываются, их невозможно различить, хотя интенсивность света обратно пропорциональна квадрату расстояния от источника, и объект, лежащий дальше от него, должен быть темнее. Как показывает опыт, большей реалистичности можно добиться при линейном затухании. В этом случае модель освещения выглядит так:

$$I = I_a k_a + \frac{I_l k_d \cos\theta}{d + K}$$

где K - произвольная постоянная.

Интенсивность зеркально отражённого света зависит от угла падения, длины волны падающего света и свойств вещества. Зеркальное отражение света является направленным. Угол отражения от идеально отражающей поверхности (зеркала) равен углу падения, в любом другом положении наблюдатель не видит зеркально отражённый свет (рис.9.1.2). Это означает, что вектор наблюдения S совпадает с вектором отражения R, а угол α равен нулю. Если поверхность не идеальна, то количество света, достигающее наблюдателя, зависит от пространственного распределения зеркального отражения света. Функция кривой отражения довольно сложна, поэтому в простых моделях освещения её обычно заменяют константой k_s, которая выбирается из эстетических соображений, либо определяется экспериментально. Таким образом

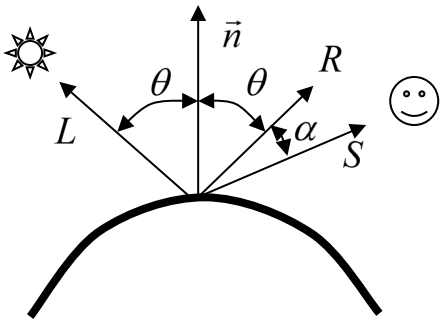


Рис.9.1.2. Зеркальное отражение

$$I = I_a k_a + \frac{I_l}{d+K}(k_d \cos\theta + k_s \cos^n \alpha) \quad ,$$

где n - степень, аппроксимирующая пространственное распределение зеркально отражённого света.

Если имеется несколько источников света, то их эффекты суммируются

$$I = I_a k_a + \sum_{j=1}^{m} \frac{I_{l_j}}{d+K}(k_d \cos\theta_j + k_s \cos^n{}_j \alpha_j) \quad .$$

Применяя формулу скалярного произведения двух векторов, запишем

$$\cos\theta = \frac{n \cdot L}{|n||L|} = n \cdot L ,$$

где n и L - единичные векторы соответственно нормали к поверхности и направления к источнику.

Точно так же

$$\cos\alpha = \frac{R \cdot S}{|R||S|} = R \cdot S ,$$

где R и S - единичные векторы, определяющие направления отражённого луча и наблюдателя. Следовательно, модель освещения для одного источника определяется как

$$I = I_a k_a + \frac{I_l}{d+K}[k_d (n \cdot L) + k_s (R \cdot S)^n] \quad .$$

9.2. **Определение нормали к поверхности**

Нормаль к поверхности представляет её локальную кривизну, а, следовательно, и направление зеркального отражения. Если известно аналитическое описание поверхности, то нормаль вычисляется непосредственно. Для многих поверхностей бывает задана лишь их полигональная аппроксимация. Зная уравнение плоскости каждой грани можно определить направление внешней нормали. В алгоритмах компьютерной графики для удаления невидимых линий и поверхностей

используются только рёбра и вершины, поэтому, чтобы объединить их с моделью освещения, необходимо знать значение нормали не только на рёбрах, но и в узловых точках.

Определение нормали в вершине V_1 рассматриваемой модели рис.9.2.1 можно представить как

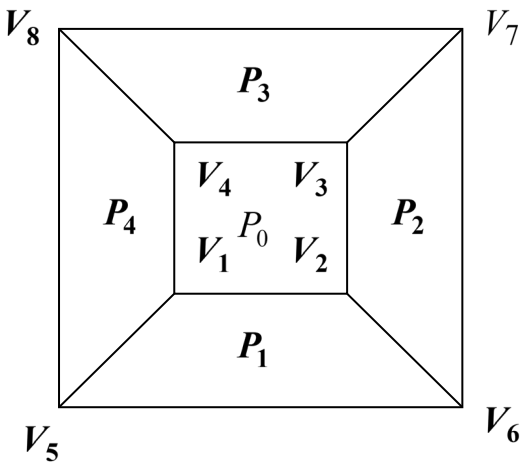


Рис.9.2.1. Аппроксимация полигональной поверхности

$$n_{V_1} = (a_0 + a_1 + a_4)i + (b_0 + b_1 + b_4)j + (c_0 + c_1 + c_4)k,$$

где $a_0, a_1, a_4, b_0, b_1, b_4, c_0, c_1, c_4$ - коэффициенты уравнений плоскостей трех многоугольников P_0, P_1, P_4, окружающих V_1.

Если же уравнения плоскостей не заданы, то нормаль к вершине можно определить, усредняя векторные произведения всех рёбер, пересекающихся в вершине

$$n_{V_1} = V_1V_2 \otimes V_1V_4 + V_1V_5 \otimes V_1V_2 + V_1V_4 \otimes V_1V_5.$$

Если вектор не нормируется, то его величина и направление зависит от количества и площади конкретных многоугольников, а также от количества и длины конкретных рёбер. Сильнее проявляется влияние многоугольников с большей площадью и более длинных рёбер.

9.3. **Метод Гуро**

Метод плоского закрашивания получает изображение, которое носит ярко выраженный полигональный характер – сразу видно, что модель состоит из отдельных плоских граней.

Методом Гуро можно получить сглаженное изображение. Для того, чтобы изобразить объект методом построчечного сканирования нужно в соответствии с моделью освещения рассчитать интенсивность каждого пикселя вдоль сканирующей строки.

Рассмотрим участок полигональной поверхности. Значение интенсивности в точке P определяется линейной интерполяцией интенсивности в точках Q и R. Для получения интенсивности в точке Q - пересечении ребра многоугольника со сканирующей строкой – нужно линейной интерполяцией интенсивностей A и B найти

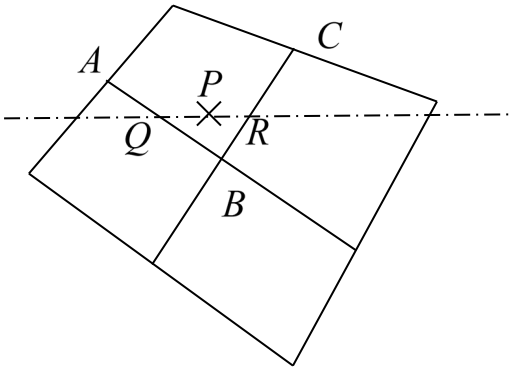


Рис.9.3.1.Интерполяция закраски

$$I_Q = uI_A + (1-u)I_B \quad 0 \le u \le 1,$$

где $u = AQ / AB$. Аналогично для получения интенсивности R линейно интерполируются интенсивности в вершинах B и C, т.е.

$$I_R = wI_B + (1-w)I_C \quad 0 \le w \le 1,$$

где $w = BR / BC$. Наконец, линейной интерполяцией по строке между Q и R находится интенсивность P, т.е.

$$I_P = tI_Q + (1-t)I_R \quad 0 \le t \le 1$$

где $t = QP / QR$.

9.4. **Закраска Фонга**

При закраске Гуро вдоль сканирующей строки интерполируется значение интенсивности, а при закраски Фонга – вектор нормали. Затем он используется в модели освещения для вычисления интенсивности пикселя. При этом достигается лучшая локальная аппроксимация кривизны поверхности и, следовательно, получается более реалистичное изображение.

При закраске Фонга аппроксимация кривизны поверхности производится сначала в вершинах многоугольников путём аппроксимации нормали в вершине (рис.9.3.1). После этого билинейной интерполяцией вычисляется нормаль в каждом пикселе. Например, снова обратясь к рисунку 9.3.1, получаем нормаль Q линейной интерполяцией между A и B, в R - между B и C и, наконец P - между Q и R. Таким образом

$$\begin{aligned} n_Q &= un_A + (1-u)n_B \quad 0 \le u \le 1 \\ n_R &= wn_B + (1-w)n_C \quad 0 \le w \le 1, \\ n_P &= tn_Q + (1-t)n_R \quad 0 \le t \le 1 \end{aligned}$$

где $u = AQ/AB, w = BR/BC, t = QP/QR$

Нормаль вдоль сканирующей строки опять можно выразить через приращение, т.е.

$$n_{P_2} = n_{P_1} + (n_Q - n_R)(t_2 - t_1) = n_{P_1} + \Delta n \cdot \Delta t,$$

где индексы 1 и 2 указывают на расположение пикселов в строке.

ЛИТЕРАТУРА

1. Н.А. Литвиненко Технология программирования на C++. Win32 API-приложения. – Спб.: БХВ-Петербург, 2010. – 288 с.; ил. – (Учебное пособие).
2. Л. Аммерал Принципы программирования в машинной графике. Пер. с англ. – М.: «СолСистем», 1992. – 224 с.: ил.
3. Роджерс Д. Алгоритмические основы машинной графики: Пер. С англ. – М.: Мир, 1989. – 512 с., ил.
4. Блинова Т.А., Пореев В.Н. Компьютерная графика / Под ред. В.Н. Порева – К.: Издательство Юниор, 2005. – 520 с., ил.
5. Херн, Дональд, Бейкер, М. Паулин. Компьютерная графика и стандарт OpenGL, 3-е издание.: Пер. с англ. – М. : Издательский дом «Вильямс», 2005. – 1168 с
6. Шишкин Е.В., Боресков А.В., Зайцев А.А. «Начала компьютерной графики». Под ред. проф. МГУ Шишкина Е.В. М: «Диалог-МИФИ», 1993 г.
7. Инженерная графика. Проецирование геометрических тел. Учебное пособие, Г. В. Буланже, И. А. Гущин, В. А. Гончарова.