Елена Козик · Светлана Хазова · Наталья Северюхина

Компьютерная графика

пособие

Учебное пособие является дополнением к лекционному курсу по дисциплине «Компьютерная графика». 1-е изд.

Учебное пособие предназначено для студентов направления -Информатика и вычислительная техника, 230400 - Информационные системы и технологии.

Palmarium
academic publishing

Imprint (only for USA, GB)
Bibliographic information published by the Deutsche Nationalbibliothek: The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Cover image: www.ingimage.com

Publisher: Palmarium Academic Publishing is a trademark of:
LAP LAMBERT Academic Publishing GmbH & Co. KG
Heinrich-Böcking-Str. 6-8, 66121 Saarbrücken, Germany
Phone +49 681 3720-310, Fax +49 681 3720-3109
Email: info@lap-publishing.com

Printed in the U.S.A.
Printed in the U.K. by (see last page)
ISBN: 978-3-8473-9245-3

Только для России и стран СНГ
Библиографическая информация, изданная Немецкой Национальной Библиотекой. Немецкая Национальная Библиотека включает данную публикацию в Немецкий Книжный Каталог; с подробными библиографическими данными можно ознакомиться в Интернете по адресу http://dnb.d-nb.de.

Изображение на обложке предоставлено: www.ingimage.com

Издатель: Palmarium Academic Publishing is a trademark of:
LAP LAMBERT Academic Publishing GmbH & Co. KG
Heinrich-Böcking-Str. 6-8, 66121 Saarbrücken, Germany
Телефон +49 681 3720-310, Факс +49 681 3720-3109
Email: info@lap-publishing.com

Напечатано в России
ISBN: 978-3-8473-9245-3

Содержание

Предисловие		4
Введение		5
Часть 1	Геометрическое моделирование	6
Глава 1	Геометрические примитивы	6
Глава 2	Геометрические модели	8
Глава 3	Однородные координаты	11
§ 1	Уравнение прямой проходящей через 2 точки	12
§ 2	Нахождение координат точки пересечения двух прямых	13
Часть 2	Двумерная графика	14
Глава 1	Двухмерные преобразования	14
Глава 2	Кривые линии	15
§ 1	Плоские кривые линии	16
§ 2	Кривые второго порядка.	19
§ 3	Кривые третьего порядка.	24
§ 4	Геометрическое сглаживание B-сплайнами	28
Часть 3	Трехмерная графика	35
Глава 1	Поверхности	36
§ 1	Построение поверхностей	37
§ 2	Поверхности вращения	38
§ 3	Поверхности плоскопараллельного переноса	39
§ 4	Линейчатые поверхности	41
§ 5	Составные (сложные) поверхности	43
§ 6	Отсеки поверхностей Кунса	46
§ 7	Отсеки поверхностей в форме Bezier	48
§ 8	Поверхности, определяемые B-сплайнами	50
Глава 2	Наглядные изображения и проецирование	51
§ 1	Параллельное проецирование	51
§ 2	Центральное проецирование (перспектива)	54
Глава 3	Тени	55
Часть 4	Работа с векторными графическими системами (AutoCAD)	65

Часть 5 Рекурсия и фракталы 67
Глава 1 Рекурсия и итерация 67
Глава 2 Графика и случайные числа 70
Глава 3 Кривые Гильберта 71
Глава 4 Фракталы и фрактальная графика 73
§ 1 Понятие размерности и ее расчет 75
§ 2 Геометрические фракталы 76
§ 3 Алгебраические фракталы 78
§ 4 Стохастические фракталы 79
§ 5 Системы итерируемых функций (IFS) 81
Часть 6 Теоретические сведения о метафайлах 82
Список использованных источников 105

Предисловие

Для плодотворного освоения такой дисциплины, как компьютерная графика, необходим определенный багаж знаний. В ходе изучения дисциплин начертательной геометрии, инженерной графики, математического анализа и программирования, будущие специалисты в сфере компьютерных технологий получают тот объем информации, который позволяет без особых проблем овладевать основами компьютерной (машинной) графики.

В результате изучения данного курса студенты должны овладеть основными понятиями компьютерной графики, ознакомиться с современным состоянием и тенденцией развития компьютерной графики, на примере практических задач компьютерной графики развитие методологического мышления студентов и приобретение ими навыков работы с графическими пакетами (на примере AutoCAD).

Они должны научиться применять алгоритмы представления, хранения и обработки графической информации, использовать знания и навыки, полученные во время изучения курсов информатики и программирования.

Студенту, обучающемуся в Вузе прививается хорошая, на наш взгляд, привычка - работать с массой информации, отбирая для себя необходимое. Но иногда поиски нужной лекции, книги или задания превращается в неоправданно долгий и мучительный процесс. Создавая этот учебный материал, авторы руководствовались принципом удобства, как для студентов, так и для преподавателей. Методическое пособие призвано облегчить выполнение лабораторных работ по компьютерной (машинной) графике.

Мы выражаем нашу признательность Павлову С.И. за предоставленную информацию и помощь в написании методического пособия. Авторы также благодарят студентов специальностей ВМК, ПОВТ оказавших посильную помощь в подготовке к изданию этого учебного материала.

Введение

Информация, представленная в визуальной форме, воспринимается легче, при этом сложные информационные структуры и взаимосвязи осознаются за более короткий промежуток времени, в большем объёме и с меньшими искажениями по сравнению с прочими используемыми методами. Людям очень трудно иметь дело с моделями явлений реального мира или абстрактных понятий без их визуального представления. Компьютерная (машинная) графика служит ценным передатчиком информации даже в случае обмена между людьми данными, полученными от ЭВМ или предназначенными для обработки на ЭВМ.

Описание, конструирование, манипулирование и представление геометрических объектов являются центральными работами в графических системах. Их поддержка в требуемом объеме за счет соответствующих математических методов, алгоритмов и программ оказывают существенное влияние на возможности и эффективность графической системы.

Современная компьютерная (машинная) графика - это тщательно разработанная дисциплина. Обстоятельно исследованы сегменты геометрических преобразований и описаний кривых и поверхностей. Также изучены, но все еще продолжают развиваться методы растрового сканирования, отсечение, удаление линий и поверхностей, цвет, закраска, текстура и эффекты прозрачности. Сейчас наибольший интерес представляют именно эти разделы компьютерной графики.

Каждая глава представленного методического пособия представляет собой теоретический материал, необходимый для выполнения лабораторных работ и расчетно-графического задания.

Часть 1 Геометрическое моделирование

Глава 1 Геометрические примитивы

В машинной графике используется понятие геометрического примитива.

Графический примитив - простейший геометрический объект, отображаемый на экране дисплея или на рабочем поле графопостроителя: точка,

отрезок прямой, дуга окружности или эллипса, прямоугольник и т.п…

К примитивам относят точку, отрезок прямой (в дальнейшем прямую) и окружность. В большинстве случаев точку рассматривают, как окружность с нулевым радиусом. При растровом выводе графической информации изображение на экран выводится по отдельным точкам, поэтому формирование на экране геометрических примитивов сводится к табулированию уравнения отрезка прямой или дуги окружности.

Итак: Объекты могут быть заданы, как:

а) двумерные объекты.

Они моделируются при помощи таких *примитивов*, как:

отрезки (заданные двумя конечными точками); многоугольники (определенные списком вершин и, возможно, заполняющим узором); окружности (описываемые центром, радиусом и, возможно, заполняющим узором); полиномиальные кривые (заданные своими коэффициентами);

б) трехмерные объекты.

В случае трехмерных измерений соответствующие примитивы определяются путем добавления координаты **Z**.

Можно также ввести и примитивы, существующие только в трехмерном пространстве: многогранники; пирамиды; сферы; цилиндры; поверхности, описываемые некоторыми полиномиальными функциями.

Системы моделирования тел порождают трехмерные объекты, основываясь на: интерактивном задании параметров (при взаимодействии с либо авто-

номном.

При автономном задании параметры можно вносить в файлы данных, созданных другой программой, или с помощью текстового редактора. С другой стороны, можно воспользоваться процедурным описанием, аналогичным тому, которое применяется для генерации фрактальных кривых и ландшафтов. Объект может быть также смоделирован непосредственно, как твердое тело, либо опосредованно, как объем, ограниченный поверхностью. В системах, построенных на основе конструктивной геометрии сплошных тел, объекты формируются из твердых тел - примитивов, таких, как блоки, цилиндры и сферы.

Примитивы можно комбинировать с помощью трехмерных теоретико-множественных операций:

а) объединение (соединение двух объектов);

б) пересечение (выделение общего подмножества);

в) разность (взятие всего первого объекта, за исключением тех его частей, которые являются общими со вторым объектом).

Косвенное задание объектов производится в системах с граничным представлением. Оно также дает возможность выполнять теоретико-множественные операции, однако при этом объект определяется как ограниченный плоскими гранями, цилиндрическими гранями или даже участками поверхности, заданными полиномиальными функциями. Такое описание поверхностей используется аэрокосмическими и автомобилестроительными компаниями. Симметричный объект можно описать с помощью поверхности вращения. Ваза или бутылка задается своей образующей (кривой, описывающей силуэт) и осью вращения. Операция переноса аналогична движению поворота: в этом случае объем формируется

путем перемещения грани произвольной формы, включая отверстия, вдоль пространственной кривой.

Геометрические примитивы используются при построении объектов как в двухмерной (2D), так и в трехмерной графике (3D). Построение объектов при помощи геометрических примитивов в 3D получило название трехмерного геометрического моделирования.

Глава 2 Геометрические модели

Во многих приложениях машинной графики возникает потребность в представлении трехмерных тел (вычислительный эксперимент, автоматизация проектирования, роботизация, вычислительная томография, тренажеры, видеографика и т.д.).

Можно выделить две основные задачи, связанные с представлением трехмерных тел, - построение модели уже существующего объекта и синтез модели заранее не существовавшего объекта.

При решении первой задачи в общем случае может потребоваться задание бесконечного количества координат точек. Чаще же всего объект с той или иной точностью аппроксимируют некоторым конечным набором элементов, например, поверхностей, тел и т.п.

При решении второй задачи, выполняемой чаще всего в интерактивном режиме, основное требование к средствам формирования и представления модели - удобство манипулирования.

Используются три основных типа 3D моделей:

а) каркасное представление, когда тело описывается набором ребер;

б) поверхностное, когда тело описывается набором ограничивающих его поверхностей;

в) модель сплошных тел, когда тело формируется из отдельных базовых геометрических и, возможно, конструктивно - технологических объемных элементов с помощью операций объединения, пересечения, вычитания и преобразований.

Важно отметить, что 3D системы существенно ориентируются на область приложений так как многие характерные для них задачи, выполняемые программным путем, стоят очень дорого и сильно зависят от выбора возможных моделей. Типичными такими задачами, в частности, являются получение сечений и удаление невидимых частей изображения. Обычно имеется много вариантов реализации различных моделей в большей или меньшей степени эффективных в зависимости от различных областей приложений и решаемых задач. Поэтому в 3D системах стремятся использовать многообразие моделей и поддерживать средства перехода от одной модели к другой.

Другим важным обстоятельством является то, что для современных систем характерно стремление моделировать логику работы, принятую пользователем. Это требует наличия средств перехода от модели, удобной для пользователя, к модели удобной для визуализации (модели тел в виде граней).

Элементы моделей

При формировании 3D модели используются:

а) двумерные элементы (точки, прямые, отрезки прямых, окружности и их дуги, различные плоские кривые и контуры);

б) поверхности (плоскости, поверхности, представленные семейством образующих, поверхности вращения, криволинейные поверхности);

в) объемные элементы (параллелепипеды, призмы, пирамиды, конусы, произвольные многогранники и т.п.).

Из этих элементов с помощью различных операций формируется внутреннее представление модели.

Методы построения моделей

Используются два основных способа формирования геометрических элементов моделей - это построение по заданным отношениям (ограничениям) и построение с использованием преобразований.

Построение с использованием отношений

Построение с использованием отношений заключается в том, что задаются:

а) элемент подлежащий построению;

б) список отношений и элементы к которым относятся отношения.

Например, построение прямой, проходящей через точку пересечения двух других прямых и касательную к окружности.

Используется два способа реализации построения по отношениям - общий и частный.

При общем способе реализации построение по заданным отношениям можно представить в виде двух шаговой процедуры:

1) на основе заданных типов отношений, элементов и параметров строится система алгебраических уравнений;

2) решается построенная система уравнений.

Очевидное достоинство такого способа - простота расширения системы - для введения нового отношения достаточно просто написать соответствующие уравнения.

Основные проблемы такого способа заключаются в следующем:

а) построенная система уравнений может иметь несколько решений, поэтому требуется выбрать одно из них, например, в диалоговом режиме;

б) система уравнений может оказаться нелинейной, решаемой приближенными методами, что может потребовать диалога для выбора метода(ов) приближенного решения.

В связи с отмеченными проблемами общий подход реализован только в наиболее современных системах и при достаточно высоком уровне разработчиков в области вычислительной математики.

Большинство же систем реализует частный подход, первым приходящий в голову и заключающийся в том, что для каждой триады, включающей строящийся элемент, тип отношения и иные элементы, затрагиваемые отношением, пишется отдельная подпрограмма (например: построение прямой,

касательной к окружности в заданной точке). Требуемое построение осуществляется выбором из меню и тем или иным вводом требуемых данных.

Преимущества такого подхода ясны - проще писать систему. Не менее очевидны и недостатки, когда пользователю требуется использовать сильно разветвленные меню и/или запоминать мало вразумительные сокращения или пиктограммы, так как обычно число требуемых вариантов построения исчисляется сотнями. Расширение системы, реализуемое добавлением новой подпрограммы, требует ее перепроектирования, по крайней мере, в части обеспечения доступа пользователя к новым возможностям. В некотором смысле предельный пример этого подхода - система AutoCAD фирмы Autodesk. Авторы даже гордятся сложностью системы: "AutoCAD предоставляет эту крайне сложную технологию"

Понятно, что перспективы за общим подходом с разумным использованием частных решений. Вместе с тем устаревшие системы типа AutoCad скорее всего также будут продолжать использоваться в силу распространенности, сложившегося круга обученных пользователей и т.п.

Построение с использованием преобразований

Построение нового объекта с использованием преобразований заключается в следующем:

1) задается преобразуемый объект;

2) задается преобразование (это может быть обычное аффинное преобразование, определяемое матрицей, или некоторое деформирующее преобразование, например, замена одного отрезка контура ломаной);

3) выполнение преобразования; в случае аффинного преобразования для векторов всех характерных точек преобразуемого объекта выполняется умножение на матрицу; для углов вначале переходят к точкам и затем выполняют преобразование.

Глава 3 Однородные координаты

Представление положения точки на плоскости с помощью трехкомпонентного вектора можно использовать для внутреннего изменения масштаба в процессе определения координат точки. Допустим, что мы изменяем единицу измерения координат *x* и *y*. В таком случае значение третьей компоненты вектора будет равно не 1, а масштабному множителю *h* и, следовательно, мы можем записать, что:

$$x = \begin{bmatrix} hx \\ hy \\ h \end{bmatrix}. \qquad (1)$$

Координаты (*hx*, *hy*, *h*) называются *однородными координатами* точки. Основное преимущество такого представления становится очевидным при обращении к уравнениям прямых. Стандартное уравнение, определяющее прямую с угловым коэффициентом $\boldsymbol{\mu}$, имеет вид:

$$y = kx + b. \qquad (2)$$

Уравнение в такой форме нельзя использовать для описания прямых, параллельных оси *У*. С другой стороны, уравнение вида:

$$ax + by + c = 0. \qquad (3)$$

допускает описание таких прямых посредством задания *b=0*. Умножив обе части уравнения (3) на масштабный множитель *h* и задав $\xi = xh,\ \eta = yh,\ \zeta = h,$ приходим к уравнению прямой в однородных координатах:

$$a\xi + b\eta + c\zeta = 0. \qquad (4)$$

§ 1 Уравнение прямой проходящей через 2 точки

При решении на построение изображений часто возникает необходимость построения отрезков прямых через две точки с заданными координатами, а уравнение самой прямой неизвестно.

Для решения этой задачи необходимо воспользоваться свойствами однородных координат и получить искомое уравнение.

Пусть координаты точек P_1 и P_2 равны (x_1, y_1, w_1) и (x_2, y_2, w_2) соответственно (смотри рисунок 1). Точка с координатами (x, y, w) будет коллинеарна

точкам P_1 и P_2, если ее координаты линейно зависимы от координат этих точек.

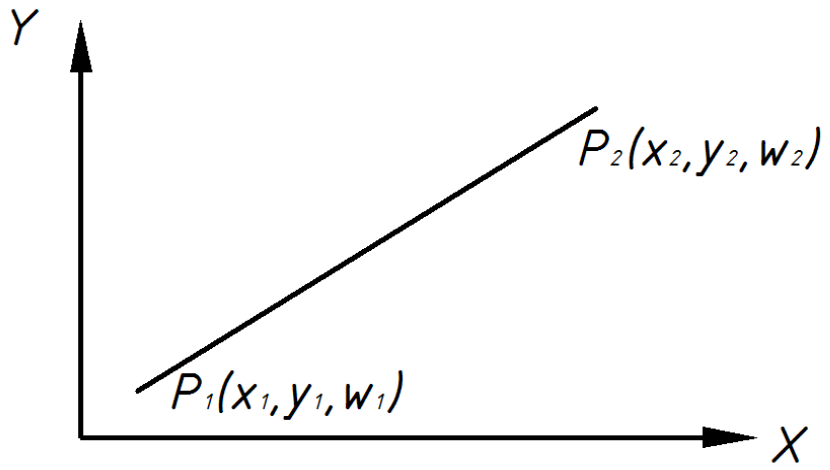


Рисунок 1 – Прямая проходящая через две точки

Это уравнение задает прямую, проходящую через точки P_1 и P_2. Раскрыв определитель, мы можем записать это уравнение в более привычном виде:

$$x(y_2w_2 - w_1y_1) + y(w_2x_2 - x_1w_1) + w(x_2y_2 - y_1x_1) = 0 \quad (5)$$

§ 2 Нахождение координат точки пересечения двух прямых

Аналогичным образом можно определить точку, лежащую на пересечении прямых L_1 и L_2 с коэффициентами (a_1, b_1, c_1) и (a_2, b_2, c_2) соответственно (смотри рисунок 2).

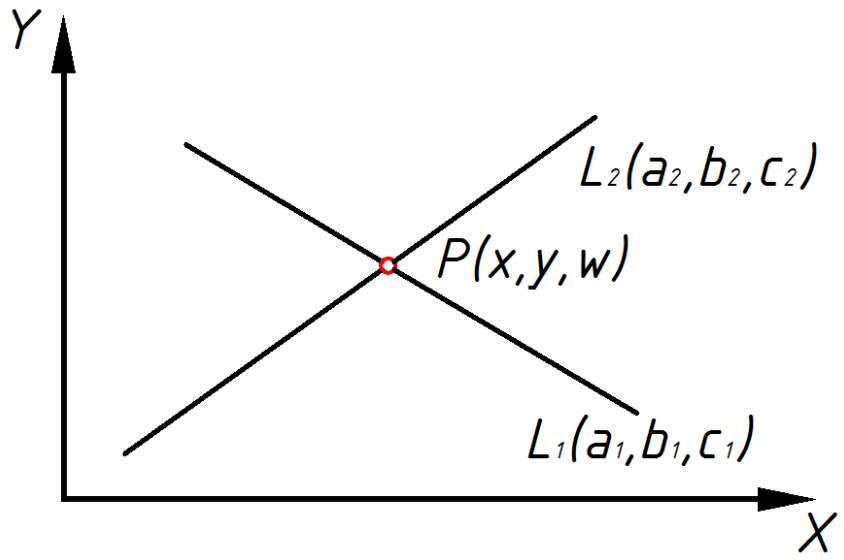


Рисунок 2 – Пересечение двух прямых

Поскольку всякая третья прямая с коэффициентами (a, b, c), проходящая через эту точку, должна быть линейно зависимой от первых двух прямых получаем:

$$a(b_2c_2 - c_1b_1) + b(c_2a_2 - a_1c_1) + c(a_2b_2 - b_1a_1) = 0. \qquad (6)$$

Поскольку (a, b, c) представляют собой коэффициенты произвольной прямой, проходящей через точку пересечения прямых L_1 и L_2, члены, стоящие в последнем уравнении в скобках, указывают координаты этой точки пересечения.

Пары уравнений (5, 6) иллюстрируют *двойственность*, существующую между точками и прямыми, задаваемыми на плоскости с однородными координатами. Набор из трех чисел может представлять как прямую, так и точку, и, следовательно, уравнения прямой, заданной двумя точками, и уравнения точки, заданной пересечением двух прямых, идентичны. Принцип двойственности широко используется в проективной геометрии, поскольку для каждого свойства точек и прямых можно получить двойственный результат с помощью взаимной замены точки и прямой.

Часть 2 Двумерная графика

Глава 1 Двухмерные преобразования

Очень распространенная задача машинной графики заключается в определении местоположения пиксела (x, y) после его поворота относительно точки (x_0, y_0) на угол Θ. Пусть r – длина вектора, соединяющего точку (x, y) с точкой (x_0, y_0). Поскольку ее значение при повороте не изменяется, справедливы следующие уравнения (обозначения расшифровываются на рисунке 3).

Разложив косинус и синус суммы двух углов в уравнении осуществив подстановку выражений для тригонометрических функций получаем уравнения, определяющие значения (X, Y) новых координат через старые:

$$X = x_0 + (x - x_0)\cos\theta - (y - y_0)\sin\theta, \qquad (7)$$

$$Y = y_0 + (x - x_0)\sin\theta + (y - y_0)\cos\theta.$$

Перенос точки можно легко учесть, прибавив к значениям соответствующую координат величину перемещения. Изменение масштаба производится при помощи умножения значений координат на масштабный множитель.

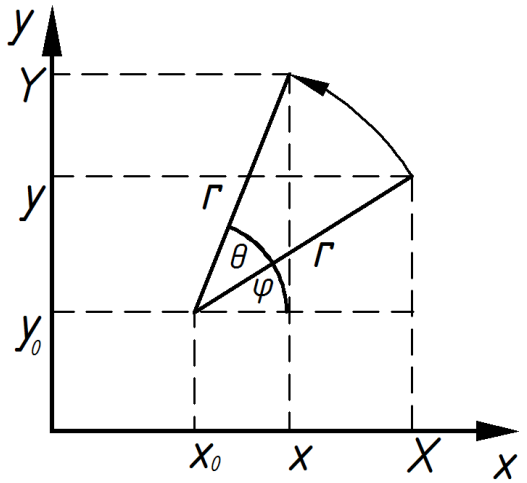


Рисунок 3 – Расшифровка обозначений, входящих в выражение, связывающее новые координаты X, Y со старыми x, y при повороте пикселя на угол Θ относительно точки (x_0, y_0)

Глава 2 Кривые линии

Одним из наиболее употребительных объектов в компьютерной графике является линия. В расширенном Евклидовом пространстве E_3+ кривые линии традиционно рассматриваются как результат пресечения двух поверхностей (смотри рисунок 4).

Параметрическое представление кривых выбирается по целому ряду причин, в том числе потому, что зачастую объекты могут иметь вертикальные касательные. При этом аппроксимация кривой y = f(x) аналитическими функциями была бы невозможной. Кроме того, кривые, которые надо представлять, могут быть неплоскими и незамкнутыми. Наконец, параметрическое представление обеспечивает независимость представления от выбора системы координат и

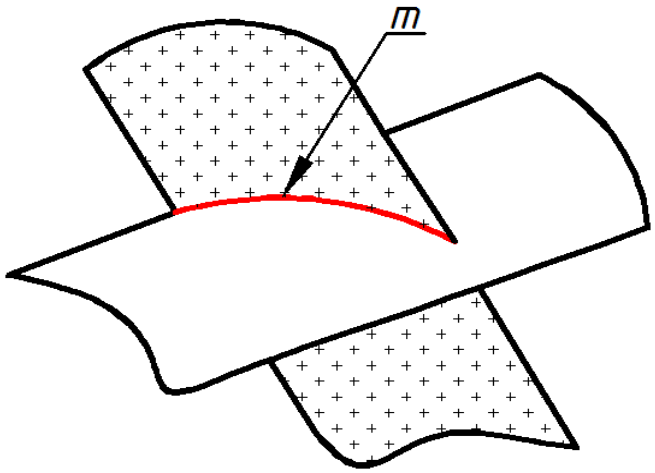


Рисунок 4 – Пересечение двух поверхностей

соответствует процессу их отображения на устройствах: позиция естественным образом определяется как две функции времени x(t) и y(t).

Различают плоские и пространственные кривые. Если в качестве одной из поверхностей выступает плоскость, то говорят о плоской кривой, во всех остальных случаях получаются пространственные кривые.

Проектирование обводов сложных конструктивных форм напрямую связано с вопросом формирования кривых по заданным условиям.

В практической работе проектировщику приходится иметь дело с двумя большими классами кривых: представляющих дуги простых кривых (графиков функций) и составных (сложных). Составные кривые (обводы) конструируются из ряда дуг простых.

Форма и характер поведения кривой в окрестности любой точки определяется ее дифференциальными характеристиками.

§ 1 Плоские кривые линии

Важное значение при формировании как 2D, так и 3D моделей имеет построение элементарных кривых.

Кривые в трехмерном пространстве с точки зрения синтетической начертательной геометрии, рассматриваются как пересечение двух поверхностей (смотри рисунок 4). Различают плоские и пространственные кривые. Если в качестве одной из поверхностей выступает плоскость, то говорят о плоской кривой, во всех остальных случаях получаются пространственные кривые.

В инженерной графике принято рассматривать кривую линию кинемтически, то есть как траекторию, описанную непрерывно движущейся точкой. Линии могут быть трансцендентными и алгебраическими, в зависимости от типа уравнений, которыми они описываются. Форма и характер поведения кривой в окрестности любой точки определяется ее дифференциальными характеристиками касательной t, нормалью n и кривизной ρ, которые определяются в соответствии с рисунком 5.

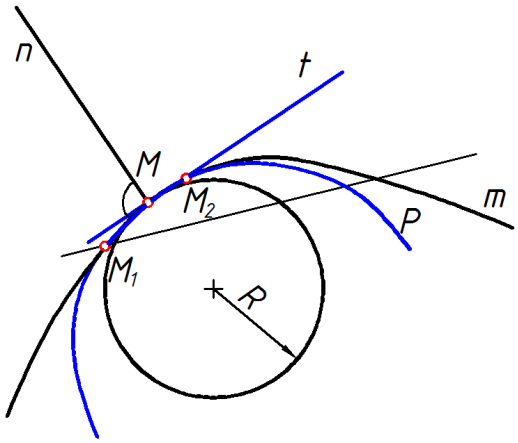


Рисунок 5 – Кинематический способ построения кривых

Касательной t к кривой m является секущая M_1M, находящаяся в предельном положении, когда M_1 стремится к M, или $t=lim\ M_1M$.

Нормаль n в заданной точке M - это линия перпендикулярная касательной t в той же точке. Кривизна определяется, как величина обратная радиусу круга кривизны $\rho = \frac{1}{R}$. А круг кривизны - это предельное положение окружности, проходящей через точки M_1, M и M_2, при одновременном бесконечном стремлении точек M_1 и M_2 к точке M.

Основной характеристикой обвода является гладкость. Под гладкостью понимают число совпавших производных (уравнений стыкующихся кривых) в точках стыка. Это означает, что если при построении обвода, у дуг разных кривых совпадают касательные, то обвод первого порядка гладкости; если совпадают еще и нормали – второй порядок гладкости; при совпадении кривизны – третий порядок гладкости.

Кривые строятся, в основном, следующими способами: той или иной интерполяцией по точкам; вычислением конических сечений; расчетом пересечения поверхностей; выполнением преобразования некоторой кривой; формированием замкнутых или разомкнутых контуров из отдельных сегментов, например, отрезков прямых, дуг конических сечений или произвольных кривых.

В качестве последних обычно используются параметрические кубические кривые, так как это наименьшая степень, при которой обеспечиваются: непрерывность значения первой (второй) производной в точках сшивки сегментов кривых; возможность задания неплоских кривых.

Для описания различных фигур на плоскости, образованных замкнутыми кривыми линиями, которым отчасти присуща осевая симметрия, довольно часто используется система полярных координат, показанная на рисунке 6, в которой координаты точки *P* задаются радиусом *r = OP* и углом *q = XOP*.

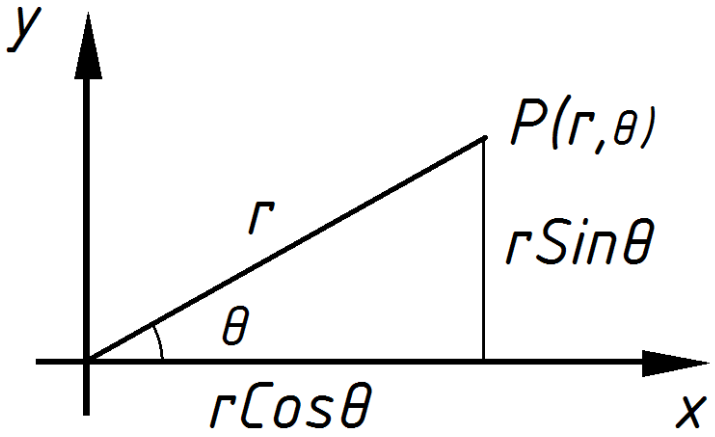


Рисунок 6 – Система полярных координат

Полярные координаты и декартовы координаты связаны соотношениями:

$$\begin{cases} x = r \cdot Cosq, \\ y = r \cdot Sinq. \end{cases} \tag{8}$$

Уравнение кривой в полярных координатах связывает r и q. Если уравнение в полярных координатах имеет явный вид r = r(q), то уравнения:

$$\begin{cases} x = r(q) \cdot Cosq, \\ y = r(q) \cdot Sinq. \end{cases} \tag{9}$$

суть параметрические уравнения этой кривой.

Уравнения вида: $r = a + bCos(nq)$, представляющие замкнутую кривую с n выступами, симметрично расположенными по окружности (смотри рисунок 7) позволяют описывать в полярных координатах различные профили. Кривые

такого вида применяются не только для простых кулачков; на их основе можно проектировать поперечные сечения тел, у которых коэффициенты *a* и *b* меняются вдоль оси, перпендикулярной плоскости OXY.

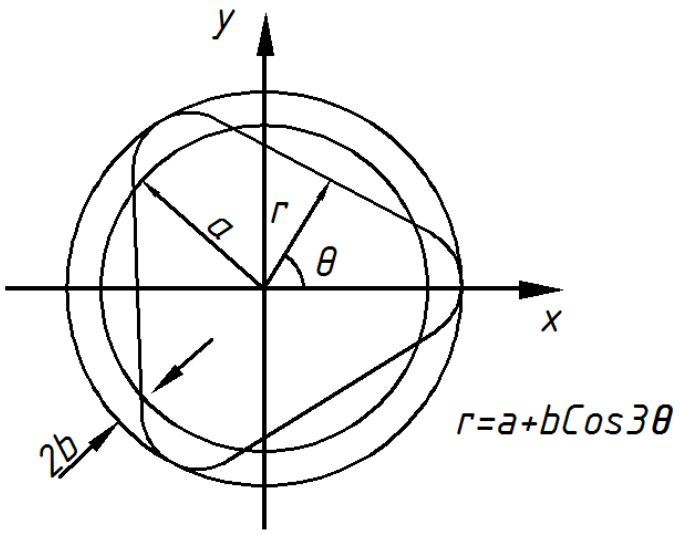


Рисунок 7 – Замкнутая кривая с *n* выступами

§ 2 Кривые второго порядка

К этому классу кривых относятся параболы, гиперболы, эллипсы, окружности, то есть все линии, уравнения которых содержат степени не выше второй. Кривая второго порядка не имеет *точек перегиба.* Прямые линии являются всего лишь частным случаем кривых второго порядка. Формула кривой второго порядка в общем виде может выглядеть, например, так:

$$x^2 + a_1 y + a_2 xy + a_3 x + a_4 y + a_5 = 0. \qquad (10)$$

Таким образом, для описания бесконечной кривой второго порядка достаточно пяти параметров. Если требуется построить отрезок кривой, понадобятся еще два параметра.

Полярные координаты имеют целый ряд недостатков:

- отсутствие простой связи между системами полярных координат с различными точками отсчета;

- описание касательных и нормалей в полярных координатах осуществляется непросто, поэтому касательные и нормали следует определять при помощи параметрических уравнений в декартовых координатах;

- полярный угол *q* точки *P (x, y)* находится только при помощи обратных тригонометрических функций.

Традиционно кривые второго порядка рассматриваются, как результат пересечения прямого кругового конуса с плоскостью. Особенностью таких кривых является то, что они не несут на себе особых точек. Общее уравнение коники имеет вид:

$$Ax^2 + By^2 + Cxy + Dx + Ey + F = 0. \qquad (11)$$

при этом вид конкретного сечения определится набором значений параметров *A, B, C, D, E* и *F*.

Если касательная, в точке $1(x_1,y_1)$ конического сечения, образует с осью *OX* угол Θ то ее уравнение можно представить, как:

$$(Ax_1 + Cy_1 + D)Cos\theta + (Cy_1 + By_1 + E)Sin\theta = 0. \qquad (12)$$

Наложив пять независимых условий подобного рода, мы получаем систему из пяти линейных уравнений для отношений *A:B:C:D:E:F*. Чтобы избавиться от необходимости решения этих уравнений, Лайминг пользуется следующими известными классическими приемами, демонстрирующими некоторые преимущества уравнений неявного вида.

Прежде всего заметим, что если два конических сечения задаются уравнениями $S_1(x, y) = 0$ и $S_2(x, y) = 0$ (или для краткости $S_1 = 0$ и $S_2 = 0$), то уравнение:

$$(1 - \lambda) \cdot S_1 + \lambda \cdot S_2 = 0. \qquad (13)$$

удовлетворяется как для точек, принадлежащих $S_1 = 0$, так и для точек $S_2 = 0$. Уравнение (13) в этом случае (поскольку оно второго порядка) определяет еще одно коническое сечение, проходящее через точки пересечения кривых $S_1 = 0$ и $S_2 = 0$, при любых значениях λ (смотри рисунок 8). При варьировании величины λ образуется семейство (или пучок) конических сечений, два из которых определяются уравнениями $S_1 = 0$ (при $\lambda = 0$) и $S_2 = 0$ (при $\lambda = 1$).

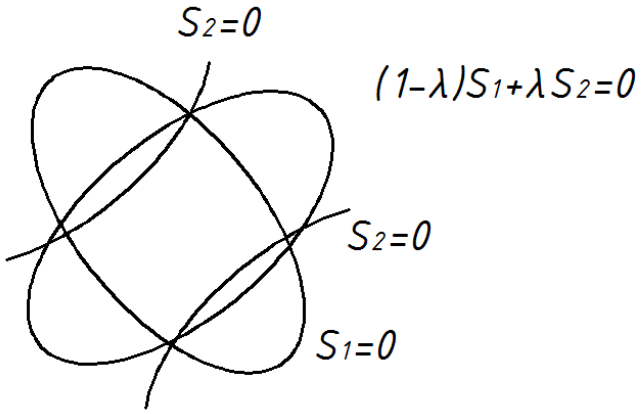


Рисунок 8 – Семейство конических сечений

для определения величины параметра l в общем случае необходимо задать еще одну точку (которая не является точкой пересечения), лежащую на кривой $(1 - \lambda)S_1 + \lambda S_2 = 0$. Если этой точкой является точка $P_1(x_1, y_1)$, то:

$$\lambda = \frac{S_1(x_1, y_1)}{(S_1(x_1, y_1) - S_2(x_2, y_2))}. \qquad (14)$$

Заметим далее, что уравнение $(a_1 \cdot x + b_1 \cdot y + c_1) \cdot (a_2 \cdot x + b_2 \cdot y + c_2) = 0$, или $l_1 \cdot l_2 = 0$, является уравнением второго порядка, которому удовлетворяют все точки, лежащие на паре прямых $l_1 = 0$ и $l_2 = 0$. Это уравнение фактически определяет вырожденное коническое сечение, получаемое в результате рассечения конуса плоскостью, проходящей через его вершину параллельно оси. Такую пару прямых можно использовать для определения невырожденных конических сечений при помощи уравнения (13).

Итак, мы видим, что уравнение $(1 - \lambda) \cdot l_1 \cdot l_2 + l \cdot l_3 \cdot l_4 = 0$ представляет семейство, или «пучок» конических сечений, проходящих через четыре точки пересечения двух пар прямых линий (l_1, l_2) и (l_3, l_4) (смотри рисунок 9). Задавая произвольную пятую точку, можно определить значение λ.

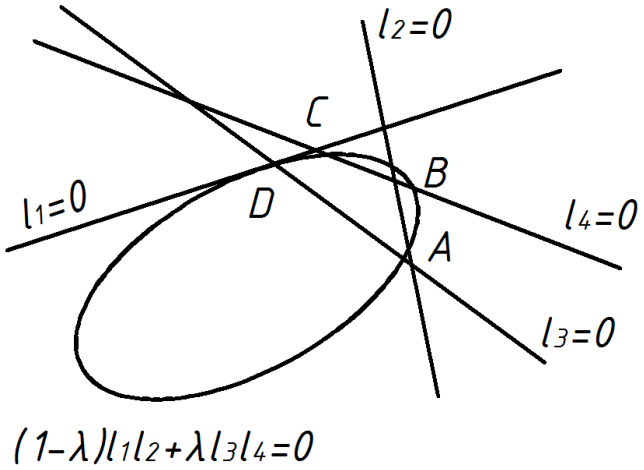


Рисунок 9 – «Пучок» конических сечений, проходящих через четыре точки пересечения

Используя этот метод, можно найти коническое сечение, имеющее две заданные касательные в двух точках и проходящее через третью заданную точку. Из рисунка 9 видно, что по мере продвижения точки C к точке D хорда CD стремится к касательной (в точке D) к изображенному на рисунке коническому сечению. Подобным же образом, если B сдвигается к A, хорда AB стремится к касательной к этому сечению в точке A. Следовательно, если прямые $l_3 = 0$ и

$l_4 = 0$ совпадают, то уравнение $(1-\lambda)\cdot l_1 \cdot l_2 + l \cdot l_3 \cdot l_4 = 0$ представляет пучок конических сечений, проходящих через точки A и D, причем l_1 является касательной в точке A, а l_2 — касательной в точке D. Выбор третьей точки F определяет параметр λ.

Коническое сечение в данном случае определяется четырьмя точками: двумя точками касания A и D, точкой пересечения касательных E и некоторой четвертой точкой F, так называемой опорной точкой. Если F выбрана внутри треугольника AED, коническое сечение всегда образует непрерывную кривую между точками A и D, проходящую внутри треугольника AED. Если F делит пополам прямую, соединяющую середины отрезков DE и AE, то коническое сечение является параболой, известной под названием пропорциональной кривой. Если F находится между этой параболой и прямой AD, то мы получаем

эллипс. Если же эта точка находится за пределами параболы, то мы получаем гиперболу.

В предложенном Лаймингом методе проектирования поперечного сечения фюзеляжа самолета конфигурация каждого сечения определяется пятью точками (смотри рисунок 10).

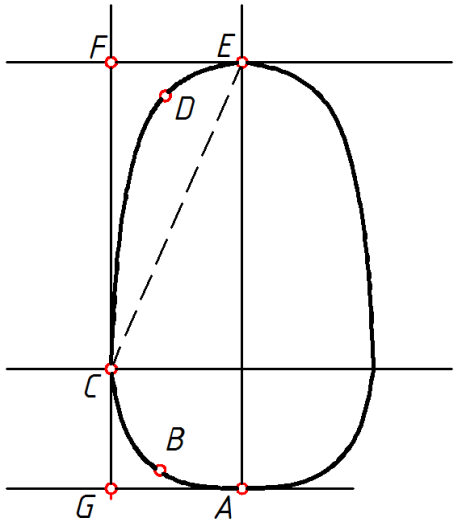


Рисунок 10 – Коническое сечение, проходящее через пять точек

Поскольку каждое сечение симметрично относительно вертикальной оси, мы должны проектировать только половину сечения, причем касательные в точках *E* и *A* должны быть горизонтальными прямыми. Поскольку точка *C* является точкой максимальной ширины сечения, касательная в С должна иметь вертикальное направление. Проектируемое сечение состоит из двух конических сечений с общей касательной в точке *C*. Фюзеляж в целом описывается пятью кривыми-следами точек *A, B, C, D* и *E*, которые эти точки оставляют при движении плоскости поперечного сечения вдоль оси Z.

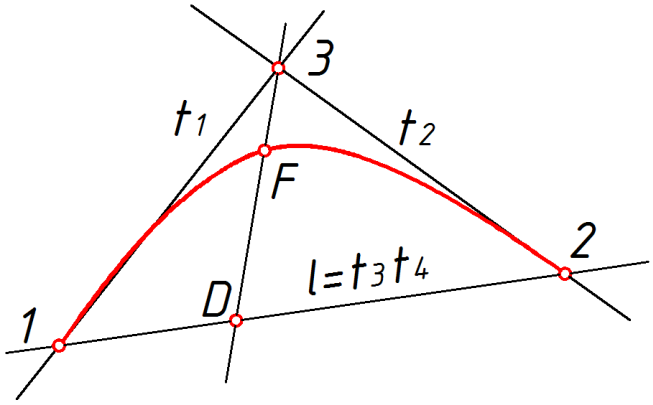


Рисунок 11 – Графическое определение инженерного дискриминанта

Если точка F выбрана внутри треугольника, образованного хордой и касательными, то коническое сечение всегда представляется непрерывной кривой. Положение точки F на медиане угла *1-3-2* (смотри рисунок 11) определит вид сечения, отношение же $f = FD/3D$ получило название инженерного дискриминанта. При значении $f = 0.5$ задается дуга параболы, при $f > 0.5$ дуга гиперболы, а при $f < 0.5$ дуга эллипса.

§ 3 Кривые третьего порядка

Отличие этих кривых от кривых второго порядка состоит в возможном наличии точки перегиба. Например, график функции $y=x^3$ имеет точку перегиба в начале координат (смотри рисунок 12). Именно

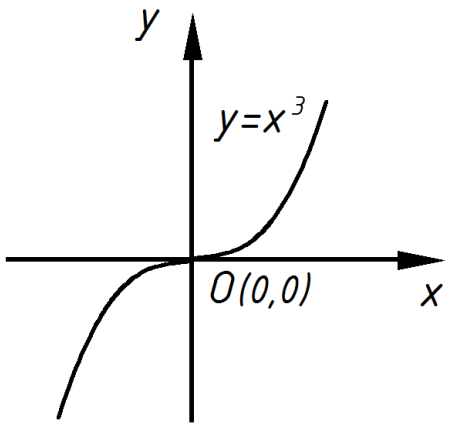


Рисунок 12 – График кривой третьего порядка

эта особенность позволяет сделать кривые третьего порядка основой отображения природных объектов в векторной графике. Например, линии изгиба человеческого тела весьма близки к кривым третьего порядка.

Все кривые второго порядка, как и прямые, являются частными случаями кривых третьего порядка.

В общем случае уравнение кривой третьего порядка можно записать так:

$$x + a_1y^3 + a_2x^2y + a_3xy^2 + a_4x^2 + a_5y^2 + a_6xy + a_7x + a_8y + a_9 = 0. \qquad (15)$$

Таким образом, кривая третьего порядка описывается девятью параметрами.

В общем виде параметрические кубические кривые можно представить в форме:

$$\begin{cases} x(t) = A_{11}t^3 + A_{12}t^2 + A_{13}t + A_{14} \\ y(t) = A_{21}t^3 + A_{22}t^2 + A_{23}t + A_{24}. \\ z(t) = A_{31}t^3 + A_{32}t^2 + A_{33}t + A_{34} \end{cases} \qquad (16)$$

где параметр t можно считать изменяющимся в диапазоне от 0 до 1, так как интересуют конечные отрезки.

Существует много методов описания параметрических кубических кривых. К наиболее применяемым относятся:

- метод Безье, широко используемый в интерактивных приложениях; в нем задаются положения конечных точек кривой, а значения первой производной задаются неявно с помощью двух других точек, обычно не лежащих на кривой;

- метод B-сплайнов, при котором конечные точки не лежат на кривой и на концах сегментов обеспечивается непрерывность первой и второй производных.

В форме Безье кривая в общем случае задается в виде полинома Бернштейна:

$$P(t) = \sum_{i=0^n} Cm_i t^i (1-t)^{m-1} \cdot P_i. \qquad (17)$$

где P_i - значения координат в вершинах ломаной, используемой в качестве управляющей ломаной для кривой, t – параметр, C_m^i - обозначает число сочетаний из m объектов по i:

$$C_{mi} = \frac{m!}{i!(m-1)!}, \tag{18}$$

Это формула неудобна для проведения вычислений – значение C_m^i лучше определять, пользуясь рекуррентной формулой:

$$C_m^i = C_{m-1}^i + C_{m-1}^{i-1}. \tag{19}$$

Задачи построения кривых по точкам возникают в проектировании и промышленном производстве, а также в машинной графике, обработке изображений и распознавании образов.

Этот класс многочленов применяется в интерактивных системах машинной графики для приближенного решения задач на построение кривых по точкам. Вместо непосредственного использования точек, представляющих обрабатываемые данные, для задания многочлена при построении искомой кривой в интерактивном режиме определяется множество точек-ориентиров. Многочлены при этом задаются не в явном виде как $y = p(x)$, а в параметрической форме:

$$\begin{cases} x = p_x(t), \\ y = p_y(t). \end{cases} \tag{20}$$

Если $(x_0, y_0), (x_1, y_1), ..., (x_m, y_m)$ - указанные точки-ориентиры, то соответствующий многочлен Безье определяется как:

$$\begin{cases} p_x(t) = \sum_{i=0}^{m} C_m^i t^i (1-t)^{m-i} x_i, \\ p_y(t) = \sum_{i=0}^{m} C_m^i t^i (1-t)^{m-1} y_i. \end{cases} \tag{21}$$

При решении многих прикладных задач, предусматривающих построение кривых по точкам, может потребоваться внести изменения в одну часть кривой, не затрагивая другие. Схему можно считать локальной, если локальные

изменения не распространяются на другие части изображения. Изменение расположения или кратности одной из точек-ориентиров требует пересчета всей кривой, даже, несмотря на то, что указанные изменения могут слабо влиять при удалении от соответствующей точки-ориентира. Кусочно-полиномиальные функции, несомненно, являются средством реализации локальных изменений. Сначала рассмотрим такие функции применительно к представлению кривой В виде $y = y(x)$, а затем — применительно к параметрическому представлению. В общем виде кусочно-полиномиальные функций представляются следующим образом:

$$p(x) = p_i(x), \quad x_i \le x \le x_{i+1}, \quad i = 0,1,\ldots,k-1; \tag{22}$$

$$p_i^{(j)}(x_i) = p_{i+1}^{(j)}(x_i), \quad j = 0,1,\ldots,r-1;$$

$$i = 1,\ldots,k-1.$$

Иногда отсутствие ограничений указывается заданием $r = 0$. При $r = 1$ речь идет о непрерывной функции, на производные которой никаких ограничений не наложено. Если $r = m+1$, то сегмент $[a, b]$ покрывается одним многочленом, и, следовательно, $r = m$ - максимальное число ограничений, порождающих нетривиальную кусочно-полиномиальную функцию. Случай, когда $r = 3$ и $m = 3$, имеет особое историческое и практическое значение, поскольку именно для обозначения соответствующих кусочно-полиномиальных функций был впервые предложен термин «сплайн».

Основной задачей машинной графики является воспроизведение изображений на экране, исходя из их математических описаний. Для этого требуется создание (обычно с помощью главной ЭВМ) файла воспроизведения визуального отображения, который затем передается в устройство отображения графической информации. В векторной графике для управления устройствами отображения требуются команды типа «провести вектор», и поэтому содержимое файла воспроизведения визуальных отображений составляют главным образом элементарные команды. В растровой графике команды типа «провести вектор» не относятся к категории элементарных и устройство

отображения преобразовывает в соответствующие конфигурации пикселов, хранящиеся в памяти обновления изображения.

Набор команд не определяет, является ли заданная вершина (x_0, y_0) верхней левой, нижней левой, верхней правой или нижней правой. Ответ определяется типом графического дисплея! В некоторых из них предполагается, что начало координат (0, 0) помещается в нижнем левом углу, а в других – в верхнем правом.

При этом крайние точки управляющей ломаной и кривой совпадают, а наклоны первого и последнего звеньев ломаной совпадают с наклоном кривой в соответствующих точках.

Используются и многие другие методы, например, метод Эрмита, при котором задаются положения конечных точек кривой и значения первой производной в них.

Общее в упомянутых подходах состоит в том, что искомая кривая строится с использованием набора управляющих точек.

§ 4 Геометрическое сглаживание B-сплайнами

B-сплайн – это сплайн, равный нулю на всех подсегментах, за исключением *m*+1.

B-сплайны можно использовать для порождения кривых аналогично тому, как это делалось с помощью многочленов Безье. Если P_i *(i = 0, 1, ..., k)* – множество точек-ориентиров, то сплайн можно определить следующим образом:

$$P(t) = \sum_{i=0}^{k} P_i N_{i,m}(t). \qquad (23)$$

где P_i - значения координат в вершинах ломаной, используемой в качестве управляющей ломаной для кривой, t - параметр, N_{im} - весовые функции, определяемые рекуррентным соотношением:

$$\begin{cases} N_{i,1} = 1, & \text{если } x_i \le t \le x_{i+1} \\ 0, & \text{в других случаях,} \end{cases}$$

$$N_{i,k}(t) = \frac{(t - x_i)N_{i,k-1}(t)}{x_{i+k-1} - x_i} + \frac{(x_{i+k} - t)N_{i+1,k-1}(t)}{x_{i+k} - x_{i+1}}. \qquad (24)$$

Диапазон значений *t* больше не должен ограничиваться сегментом [0, 1] и точки склеивания $t_1, t_2, \ldots, t_{k-1}$ должны задаваться несколько иначе, чем это было в случае многочленов Безье. Очевидно, что *P(t)* представляет собой сумму векторов, умноженную на числа (*B*-сплайны), суммы которых в свою очередь равна единице.

Наличие кратных точек вынуждает синтезируемую кривую проходить ближе к точкам-ориентирам, во многом аналогично тому, как это происходит в случае многочленов Безье.

Любой более или менее сложный чертеж состоит не только из отрезков прямых линий, окружностей и их дуг, но также и из набора кривых линий. Гладкие кривые удобно строить при помощи метода сглаживания кривой типа B-сплайна. B-сплайн- это гладкая кривая или, точнее, кривая с непрерывными старшими производными до *n*-ой, где *n* - порядок сплайна. Заметим, что линия, составленная из B-сплайнов, не будет проходить точно через заданные точки. Подобную кривую составляют из дуг полиномов третьей степени, так как такой полином обеспечивает необходимую непрерывность. Построение линии происходит с помощью итерационной процедуры.

Рассмотрим построение кубического сплайна. Пусть нам даны две соседние точки, через которые проведем кубический полином, но у полинома-4 коэффициента, следовательно нужно еще два дополнительных условия или точки. Для этого прихватим еще две соседние точки. Чем более плавной мы хотим видеть линию, тем сложнее пройти точно через точки. Если в формуле $x = q^3$, то достаточно плавности 3.

Гладкость диктуется физическими задачами, и здесь часто приходится искать компромисс между гладкостью и точностью. Например, гидродинамика

работает с поверхностями, которые описываются уравнениями четвертой степени (такой высокий порядок необходим, чтобы повысить гладкость различного рода физических устройств, рассчитанных с помощью этих уравнений, и таким образом избежать завихрений). Но с повышением порядка (то есть гладкости) сплайна точность уменьшается.

Рассмотрим рисунок 13. Пусть t - параметр, по которому пробегаем от точки P_i к точке P_{i+1}. При $t = 0$ мы находимся в точке P_i, при $t = 1$- в точке P_{i+1}. Если $0 \le t \le 1$, то мы находимся между P_i и P_{i+1}.

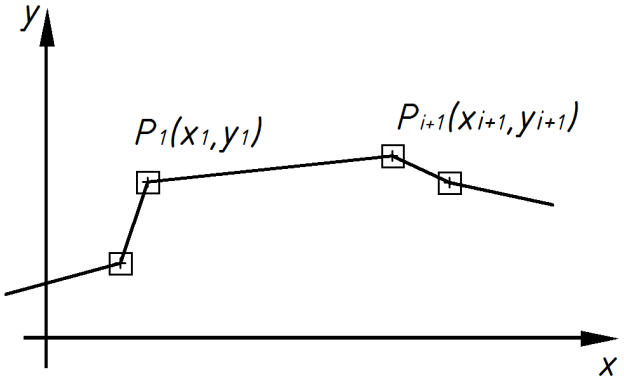


Рисунок 13 – Построение кубического сплайна

Эта линия в каждой точке имеет систему:

$$\begin{cases} x(t) = \left(\left(a_3 t + a_2\right)\cdot t + a_1\right)\cdot t + a_0, & \text{для } 0 \le t \le 1 \\ y(t) = \left(\left(b_3 t + b_2\right)\cdot t + b_1\right)\cdot t + b_0, & \text{для } 0 \le t \le 1. \\ 1 \end{cases} \qquad (25)$$

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

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

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

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

Точки b_3 - b_0 расписывают так же, но вместо X подставляют Y. Между P_i и P_{i+1} точки a и b не меняются. Если после последней точки указать первую точку, то система замкнет контур.

Достоинства B-сплайна: между точками коэффициенты постоянны; локальное изменение не влечет за собой вычисление заново всего сплайна.

Недостатки: могут возникать проблемы при аппроксимации прямой, имеющей разрывы вторых производных (например, сопряжения прямой линии и дуги окружности); с точки зрения эстетики не всегда приемлемы, так как кривизна поверхности, сконструированной с помощью сплайнов, изменяется иногда неравномерно, что приводит к искажениям (например, причудливые искажения предметов, отраженных от кузова автомобиля).

Следствия:

1. Кривые отходят от точек, точки усредняют свое влияние:

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

2. $x_{i+1}: x(1) = a_3 + a_2 + a_1 + a_0 = (x_i + 4x_{i+1} + x_{i+2})/6$

3. Изменение одной вершины ведет к изменению только четырех соседних отрезков.

4. Геометрическая интерпретация B-сплайна: так как сумма коэффициентов равна единице, а коэффициенты положительны, то мы получаем средневзвешенную точку для четырех соседей, то есть первоначальный отрезок будет находиться внутри выпуклой сплайновой оболочки. Формула расчета:

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

5. Кратность вершины усиливает ее притяжение. В острые углы линия не успевает забегать. В этом случае увеличивают кратность вершины. Порядок сплайна влияет на конфигурацию - увеличение порядка ведет к большему спрямлению (смотри рисунок 14).

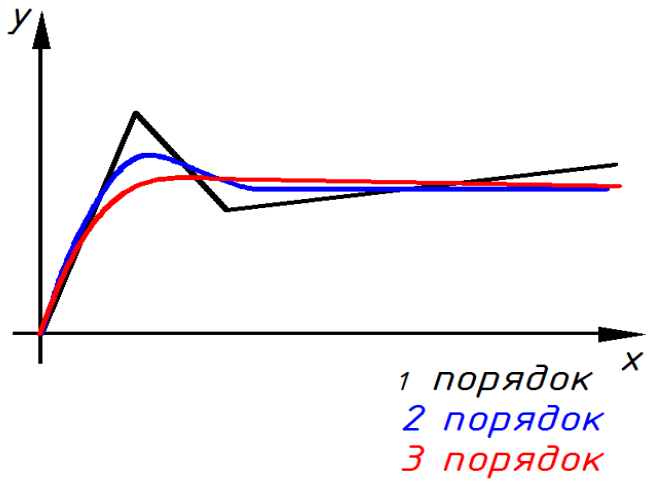


Рисунок 14 – Конфигурации сплайнов разного порядка

Математическое представление тела, составленного из простых геометрических форм (сфер, цилиндров или конусов) несложно. Но очень часто это не так; кузова автомобилей, поверхности самолетов, флюзеляжи и многое другое не так-то просто описать. Процедура, обычно используемая в этих случаях, состоит обычно в следующем:

- поверхность покрывается двумя воображаемыми группами линий; первая идет в продольном направлении, вторая - трансверсальна к первой. Эта сетка линий определяет множество ячеек, каждая из которых (в случае гладкой поверхности), будет ограничена четырьмя гладкими кривыми;
- координаты узлов этой воображаемой сетки измеряются на модели или на наборе чертежей поперечных сечений поверхности;
- с помощью интерполяции (усреднения) математически описываются эти две группы линий, образующие сетку.

Можно строить достаточно гладкие кривые и поверхности с использованием полиномов. Допустим, что мы хотим построить поверхность в виде графика функции $Z = Z(X, Y)$. Линия $Y = Const$ на этой поверхности будет представлена линией $Z = Z(X)$, она будет проходить через последовательность точек $(x_0, z_0), ..., (x_i, z_i), ..., (x_n, z_n)$ с $x_0 < ... < x_i < ... < x_n$. Наша цель провести через эти точки составную кривую $f(x)$, имеющую следующие свойства:

а) на каждом под интервале $x_{i-1} \le x \le x_i$, $i = 1, 2, ..., n$ функция $f(x)$ является кубическим полиномом;

б) ее первые и вторые производные непрерывны в узлах.

Полученная гладкая кривая называется кубическим сплайном. Термин "сплайн" возник по аналогии: это название чертежного инструмента - тонкой металлической линейки, которая может изгибаться так, чтобы проходить через заданные точки. Физически такая кривая минимизирует энергию внутренних напряжений. Математически - имеет минимальную среднеквадратичную кривизну, то есть она наиболее гладкая. Сплайны имеют много приложений в

конструировании криволинейных форм. Однако они имеют и некоторые ограничения:

а) локальное изменение влечет за собой вычисление заново всего сплайна;

б) могут возникать проблемы при аппроксимации прямой, имеющей разрывы вторых производных (например, сопряжения прямой линии и дуги окружности);

в) с точки зрения эстетики не всегда приемлемы, так как кривизна поверхности, сконструированной с помощью сплайнов, изменяется иногда неравномерно, что приводит к искажениям (например, причудливые искажения предметов, отраженных от кузова автомобиля).

Первое ограничение можно устранить с помощью В-сплайна. Это кубический сплайн (фундаментальный сплайн) четвертого порядка (или третьей степени). Про него говорят, что он имеет минимальный носитель (носитель- это число отрезков, на которых сплайн отличен от нуля).

Заметим, что кубический В-сплайн полностью определяется множеством узлов, на которых он определен, и только одной заданной величиной *Z*. В более общем виде В-сплайн $M_{mi}(x)$ порядка *m* (или степени *m - 1*) на данном множестве узлов везде равен нулю, кроме *m* последовательных отрезков $x_{i-m} < x < x_i$. Опять-таки $M_{mi}(x)$ определяется множеством узлов и одной величиной *Z*. Принято исключать последнюю степень свободы и фиксировать амплитуду В-сплайна некоторым стандартным образом.

Часто удобно для вычислений использовать нормализованный В-сплайн $N_{mi}(x)$, связанный с $M_{mi}(x)$ соотношением $N_{mi}(x) = (x_i - x_{i-m})M_{mi}(x)$.

Любой сплайн порядка *m* на множестве узлов $x_0, x_1, \ldots, x_n$ может быть выражен в виде линейной комбинации В-сплайнов, определенных на том же множестве узлов, расширенном *(m - 1)* дополнительными узлами на каждом из концов интервала, которые можно выбрать произвольно: $x_{-m+1}, x_{-m+2}, \ldots, x_{-1}$ и $x_{n+1}, \ldots, x_{n+m-1}$. Можно построить *m + n - 1* последовательных В-сплайнов на расширенном множестве узлов, каждый из которых отличен от нуля на *m* последовательных отрезках. Поэтому можно записать: $j(x) = Sc_i \cdot M_{mi}(x)$, где

$j(x)$ - любой сплайн степени *(m - 1)* на первоначальном множестве узлов и $M_{mi}(x)$ есть B-сплайн на расширенном множестве узлов, отличный от нуля при $x_{i-m} < x < x_i$; c_i - суть числовые коэффициенты; суммирование ведется по $i = 1, ..., m + n - 1$.

Если имеется множество векторов $r_0, r_1, ..., r_n$, то можно использовать их: $r(u) = Sr_i \cdot N_{4,i+1}(u)$ (суммирование ведется по $i = 0, ..., n$). Так как имеется *(n + 1)* векторных коэффициентов, то необходим набор из *(n + 1)* B-сплайнов. Последняя формула для $0 <= u <= n - 2$ является уравнением кривой, образованной кубическими B-сплайнами.

Свойства:

Некоторые простейшие свойства следуют из тождества $SN_{4,\ i+1} = 1$, $0 \le u \le n - 2$, $i = 0..n$. При u = 0 следует: $r(0) = N_{42}(0) \cdot (r_1 - r_0)$. Из этого следует, что если $r_0, r_1, .., r_n$ - вершины некоторой замкнутой ломанной, то кривая, построенная на основе B-сплайна, начинается в r_0 и ее касательная в этой точке имеет направление $(r_1 - r_0)$. Аналогичное утверждение верно и для другого конца. Главное преимущество этого сплайна заключается в том, что изменение одной из вершин влечет за собой изменение только четырех отрезков кривой. Далее, мы также можем построить кривую, аппроксимирующую ломанную с любым желаемым числом сторон. Отрезок сплайна всегда лежит в выпуклой оболочке (смотри рисунок 15):

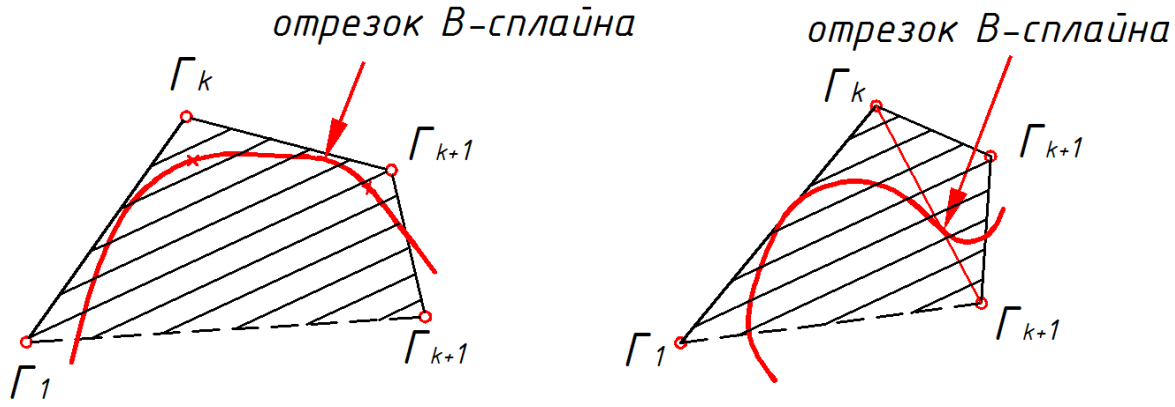


Рисунок 15 – B-сплайн 3-го порядка

Важным следствием этой выпуклой оболочки является вырождение ее в прямую линию, если четыре последовательные вершины ломанной коллинеарны, значит соответствующий сегмент кривой должен быть прямолинейным.

Имеется еще два полезных факта:

1) кривая проходит вблизи средней точки каждой стороны, за исключением 1-ой и последней точками;

2) при k = 2, ..., n - 2 кривая проходит через точки:

$$\frac{1}{6r_{k-1}} + \frac{2}{3r_k} + \frac{1}{6r_{k+1}} = \frac{2}{3r_k} + \frac{1}{3}\left(\frac{1}{2(r_{k-1} + r_{k+1})}\right)$$

Эти точки, как показано на рисунке 16 , лежат на 1/3 расстояния от r_k на прямой, соединяющей r_k с серединой отрезка между r_{k-1} и r_{k+1}.

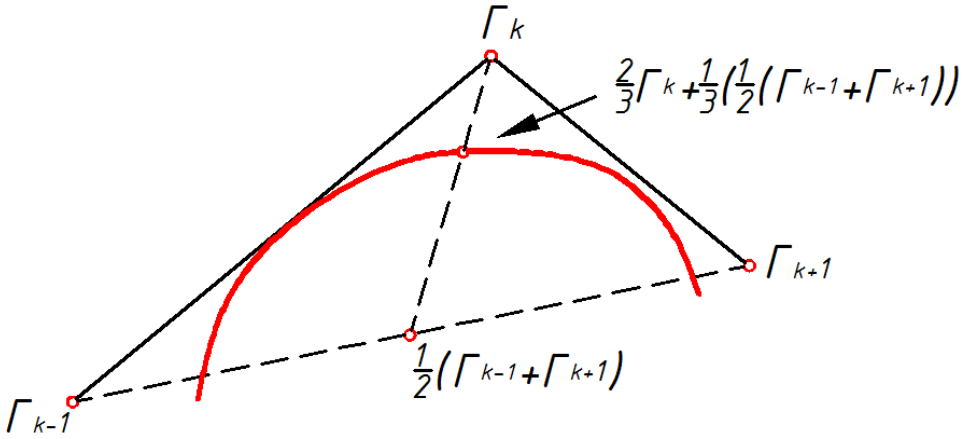


Рисунок 16 – Свойства B-сплайна

Часть 3 Трехмерная графика

Трехмерная графика нашла широкое применение в научных расчетах, инженерном проектировании, компьютерном моделировании физических объектов. В качестве примера рассмотрим сложный вариант трехмерного моделирования - создание подвижного изображения реального физического тела.

В упрощенном виде для пространственного моделирования объекта требуется:

- спроектировать и создать виртуальный каркас объекта и виртуальные материалы;

- присвоить материалы различным частям поверхности объекта;

- настроить физические параметры пространства, в котором будет действовать объект, - задать освещение, гравитацию, свойства атмосферы, свойства взаимодействующих объектов и поверхностей;

- задать траекторию движения объектов;

- рассчитать результирующую последовательность кадров;

- наложить поверхностные эффекты на итоговый анимационный ролик.

Для создания реалистической модели объекта используют геометрические примитивы (прямоугольник, куб, шар, конус и прочие) и гладкие, так называемые сплайновые поверхности.

Глава 1 Поверхности

Поверхности рассматриваются либо как двумерные множества точек, либо как одномерные множества линий. Второе определение наиболее соответствует конструированию поверхностей с использованием кинематического метода (смотри рисунок 17).

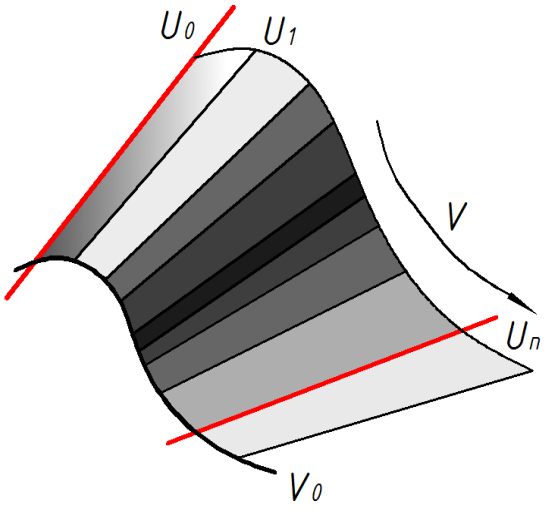


Рисунок 17 – Кинематический способ построения поверхности

Этот подход предполагает формирование поверхности в результате перемещения одной кривой *U* (образующей) по другой кривой *V* (направляющей).

В общем случае понятия направляющей и образующей чисто условные.

Перемещение кривой *V* по кривой *U* сформирует ту же самую поверхность.

Наложение условий на форму кривых и условия перемещения позволяет формировать практически любые поверхности.

В общем случае поверхность может быть определена уравнением в неявном виде F(x,y,z) = 0, явном виде z=f(x,y), или параметрическими уравнениями:

$$\begin{cases} x = X(u,v), \\ y = Y(u,v), \\ z = Z(u,v). \end{cases} \qquad (26)$$

Параметры *u* и *v* получили название криволинейных координат.

Определение условий характера перемещения образующей позволяет среди множества поверхностей выделить классы наиболее употребительных в практической деятельности.

Этот подход предполагает формирование поверхности в результате перемещения одной кривой *U* (образующей) по другой кривой *V* (направляющей).

В общем случае понятия направляющей и образующей условные. Перемещение кривой *V* по кривой *U* сформирует ту же самую поверхность.

Наложение условий на форму кривых и условия перемещения позволяет формировать практически любые поверхности.

В общем случае поверхность может быть определена уравнением в неявном виде: $F(x,y,z) = 0$, явном виде $z=f(x,y)$, или параметрическими уравнениями $x = X(u,v)$, $y= Y(u,v)$, $z = Z(u,v)$.

Параметры u и v получили название криволинейных координат.

§ 1 Построение поверхностей

Основные способы построения поверхностей:

- интерполяцией по точкам;
- перемещением образующей кривой по заданной траектории (кинематический метод);
- деформацией исходной поверхности;
- построением поверхности эквидистантной к исходной;
- кинематический принцип;
- операции добавления/удаления в структуре;
- теоретико-множественные (булевские) операции.

Широко используется бикубические параметрические куски, с помощью которых сложная криволинейная поверхность аппроксимируется набором отдельных кусков с обеспечением непрерывности значения функции и первой (второй) производной при переходе от одного куска к другому. Аналогично случаю с параметрическими кубическими кривыми, наиболее применимыми являются:

- форма Безье;
- форма B-сплайнов;
- форма Эрмита.

§ 2 Поверхности вращения

Поверхность вращения образуется в результате вращения плоской кривой

вокруг прямой, называемой осью вращения (смотри рисунок 18).

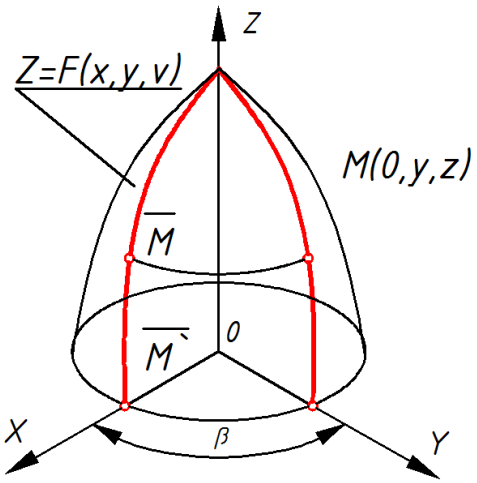


Рисунок 18 – Поверхность вращения

Уравнение поверхности вращения может быть получено из уравнения образующей, например, линии лежащей в плоскости ZoY. Для этого достаточно принять за ось вращения координатную ось OZ. Определение координат произвольной точки M образующей $z=f(y)$ при ее повороте на произвольный угол β позволит связать все три пространственные координаты.

При повороте образующей на произвольный угол любая точка M этой образующей переместится в новое положение $\overline{M}$. При этом координата z этой точки не изменится, а текущие координаты x и y с исходными координатами будут связаны соотношением:

$$Y = \sqrt{X^2 + Y^2}\ . \qquad (27)$$

Тогда уравнение, связывающее координаты перемещенной точки, может быть представлено в виде:

$$Z = f\sqrt{(X^2 + Y^2)}\ . \qquad (28)$$

Поскольку все рассуждения приведены для произвольных параметров, вывод справедлив для любой точки образующей и, как следствие, для любой точки поверхности, следовательно, полученное уравнение и есть уравнение поверхности вращения.

Задание образующей в параметрической форме:

$$r = r_m(u) = y(u)j + z(u)k, \qquad (29)$$

позволяет записать уравнение поверхности в векторной форме:

$$r = xCos\varphi(u)i + ySin\varphi(u)j + z(u)k. \qquad (30)$$

§ 3 Поверхности плоскопараллельного переноса

Еще один широкий класс поверхностей - поверхности плоскопараллельного переноса.

Поверхности этого класса получаются от перемещения образующей по направляющей при параллельности, при перемещении, последней самой себе (рисунок 19).

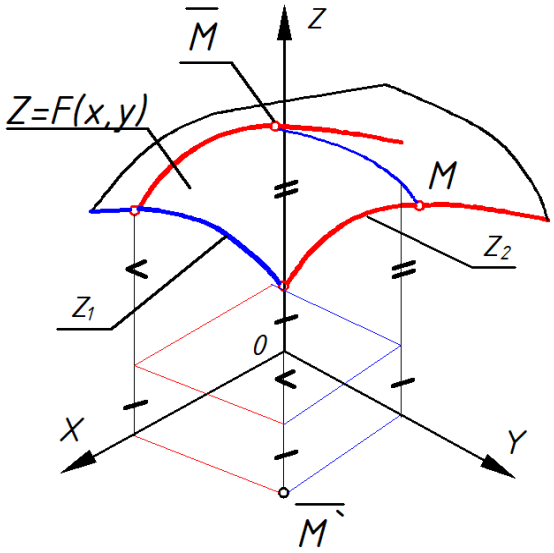


Рисунок 19 – Поверхность плоскопараллельного переноса.

Именно это и определяет способ получения уравнения такой поверхности. Пусть уравнение образующей имеет вид: $z_2 = f(y)$, а уравнение направляющей: $z_1 = t(x)$.

Тогда, после переноса на произ-вольное расстояние образующей, произвольная ее точка M переместится в

новое положение $\overline{M}$. Определим коорди- наты этой точки.

Координаты X и Y - текущие и определяются положением точки на образующей и величиной ее смещения. Координата же Z определится из условия

$$Z = Z_1 + Z_2 - Z_0. \qquad (31)$$

где Z_1 - координата точки M в составе образующей,

Z_2 - координата точки M в составе направляющей,

Z_0 – координата точки пересечения направляющей и образующей.

Приведенное выражение справедливо для любой точки образующей при любом ее перемещении и, следовательно, справедливо для любой точки конструируемой поверхности, с учетом уравнений направляющей и образующей оно будет иметь вид:

$$Z = t(x) + f(y) - Z_0. \qquad (32)$$

§ 4 Линейчатые поверхности

Широко в практике используются и поверхности, несущие на себе семейство прямых линий. Эти поверхности получили название – *линейчатые*. Эти поверхности называют еще поверхностями с тремя образующими (смотри рисунок 20).

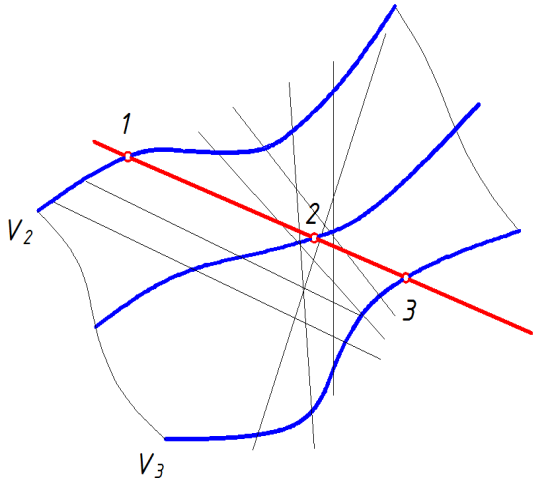


Рисунок 20 – Линейчатая поверхность

Схема образования поверхности следующая: в пространстве задаются две кривые (V_1, V_2), определяющие граничные точки отрезка прямой U, выступающей в качестве образующей. Условие прохождения семейства прямых

(образующих) определяется третьей направляющей V_3 (действительной или мнимой).

Разнообразие поверхностей, получаемых таким образом, определяется разнообразием направляющих V_1, V_2 и формой и положением направляющей V_3.

Использование отрезков прямых в качестве направляющих позволяет конструировать достаточно сложные отсеки поверхности, получившей название *косой плоскости* (смотри рисунок 21).

Ориентировка прямо-линейных образующих U_i сводится к проведению их параллельно плоскости Q, несоб-ственная прямая которой, и является третьей направляющей V_3.

Геометрическую модель такой поверхности удобнее представить в ви-

.

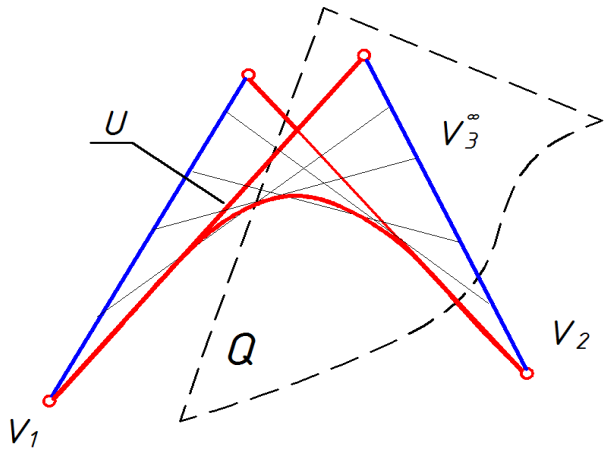


Рисунок 21 – Косая плоскость

де непрерывного каркаса

Определим направляющие V_1 *(1,2)* и V_2 *(3,4)* точками, ограничи-вающими концы отрезков, тогда их параметрические уравнения *(v = 0,...,1)* будут соответственно иметь вид:

$$X = x_1(1-v)+x_2v \quad Y = y_1(1-v)+y_2v \quad Z = z_1(1-v)+z_2v$$

и

$$X = x_3(1-v)+x_4v \quad Y = y_3(1-v)+y_4v \quad Z = z_3(1-v)+z_4v.$$

Тогда для фиксированного значения *v* координат точек S_1 и S_2 появляется возможность записать уравнение текущей образующей *(u=0,...,1)* в виде:

$$\begin{aligned} X &= x \cdot S_1(1 - u) + x \cdot S_2 u, \\ Y &= y \cdot S_1(1 - u) + y \cdot S_2 u, \\ Z &= z \cdot S_1(1 - u) + z \cdot S_2 u. \end{aligned} \qquad (33)$$

Рассмотренный способ представления может быть обобщен на более общий случай, когда направляющие представляют из себя кривые общего вида, например, кривые в форме *Bezier*.

В векторной форме линейчатые поверхности представляются как:

$$r(u, v) = r_o(v) + un(v), \qquad (34)$$

где $r(v)$ – заданная точка на прямой с параметром *v* и $n(v)$ – направляющий вектор этой прямой.

Еще одна форма представления линейчатой поверхности:

$$r(u, v) = (1-u)r_1(v) + ur_2(v), \qquad (35)$$

где $r = r_1(v)$ и $r = r_2(v)$ - направляющие линейчатой поверхности.

Особое место в ряду линейчатых поверхностей занимают *поверхности с ребром возврата (торсы)*. Это единственные изо всех простых поверхностей, допускающие построение разверток.

Линейчатая поверхность $r(u, v) = r(v) + un(v)$ будет развертываться только тогда, когда выполняется условие

$$r'(n \times n') = 0. \qquad (36)$$

Такая поверхность (торс) образуется семейством прямых, огибающая которых является кривой:

$$r = r(v) + \frac{N(r' \times n) \cdot n(v)}{N^2}, \qquad (37)$$

где $N = n \times n'$.

Беря в качестве $n(v)$ единичный касательный вектор *T* к кривой $r = r(v)$, уравнение развертывающейся поверхности можно записать в виде:

$$r = r(v) + \lambda T(v), \qquad (38)$$

или с учетом уравнений направляющих в виде

$$(r - r_1)(r' \times r_1') = 0. \qquad (39)$$

Примерами торсовых поверхностей могут служить цилиндры и конусы, имеющие вырожденное ребро возврата. Для конусов это действительная точка (вершина конуса), для цилиндра - бесконечно удаленная (точка пересечения образующих цилиндра).

§ 5 Составные (сложные) поверхности

По аналогии с обводами кривых (одномерными обводами) можно говорить об обводах поверхностей (двумерных обводах). Под двумерным обводом будем понимать составную поверхность с заданием условий на стыках.

Для этого определим параметры поверхностей, дающие возможность выполнить эти условия (дифференциальные характеристики).

Основной дифференциальной характеристикой поверхности $\boldsymbol{r} = \boldsymbol{r}(u,v)$ для произвольной точки поверхности является *нормаль*. Вектор нормали, в криволинейных координатах, может быть определен как:

$$n = \pm \frac{\left(\frac{dr}{du} \times \frac{dr}{dv} \right)}{\left| \frac{dr}{du} \times \frac{dr}{dv} \right|}. \qquad (40)$$

где производные вычисляются в точке $u = u_o$, $v=v_o$.

Плоскость ортогональная вектору нормали называется *касательной* плоскостью.

Нормаль и касательная плоскость являются дифференциальными характеристиками первого порядка.

Еще одной характеристикой (второго порядка) является кривизна. Для произвольной пространственной кривой, лежащей на поверхности, кривизна ξ может быть найдена из соотношения:

$$S^2 \xi N n = u^\tau D u, \tag{41}$$

где D - матрица второй фундаментальной формы поверхности:

$$D = \begin{vmatrix} d^2 r / du^2 & d^2 r / dudv \\ d^2 r / dudv & d^2 r / du^2 \end{vmatrix}. \tag{42}$$

Направления на поверхности, в которых значение кривизн будет достигать наибольших значений, получили названия *главных кривизн*. Произведение главных кривизн называют *полной кривизной* поверхности или *Гауссовой* кривизной $\vartheta = \xi_1 \xi_2$.

Для получения составной поверхности (двумерного обвода) необходимо добиться условия совпадения во всех точках кривой t (смотри рисунок 22), по которой производится стыковка отсеков поверхностей $F_1(x,y,z)=0$ и $F_2(x,y,z)=0$, и

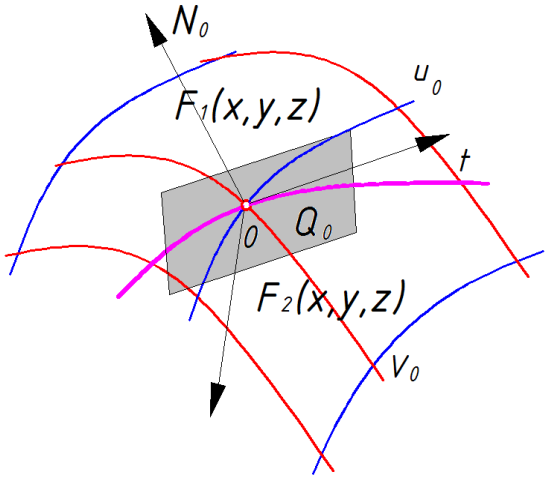


Рисунок 22 – Двумерный обвод.

касательных плоскостей Q_i. Это условие определит и совпадение касательных на кривой стыка.

При этом построенный двумерный обвод будет иметь первый порядок гладкости. Обеспечение условия совпадения главных кривизн позволит получить обвод второго порядка гладкости.

Наиболее характерным, при конструи-ровании сложных поверхностей технических форм, является представление поверхности линейчатым каркасом, сетью пространственных или плоских кривых - u_i и v_i , делящих исходную

поверхность на отдельные «куски» (отсеки ϑ_i) простых поверхностей (смотри рисунок 23).

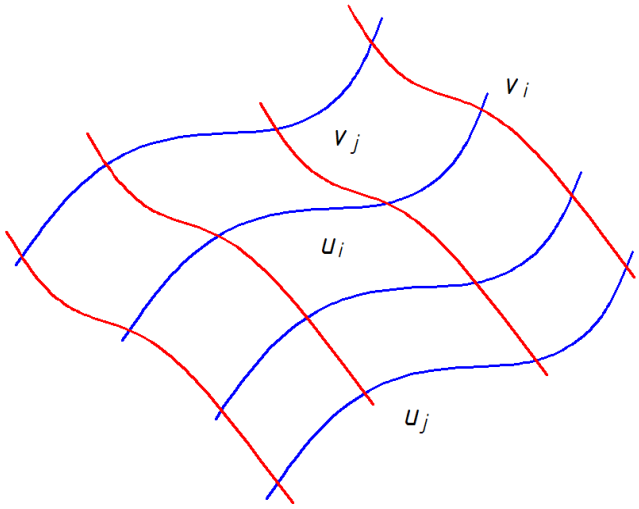


Рисунок 23 – Линейчатый каркас

При таком подходе возникает не-обходимость в конструировании элементарных отсеков поверхнос-тей ϑ_i по условиям задания границ (положению сети линий u_i и v_i) и склейки их в единое целое.

Наиболее употребительные из нихэто отсеки поверхности Кунса, Безъе и сплайновые поверхности.

§ 6 Отсеки поверхностей Кунса

Пусть сетка кривых уже построена каким-либо образом; параметры этих кривых определятся как u и v соответственно. (смотри рисунок 24)

В этом случае каждый из отсеков будет ограничен парой u-кривых и парой v-кривых. Допустим, что u и v изменяются вдоль соответствующих границ в пределах $0,\ldots,1$. Тогда задача определения поверхности может быть сведена к нахождению функции $r=r(u,v)$ с подходящим типом «хорошего поведения», которая при $u=0, u=1, v=0$ или $v=1$ представляет нужную граничную кривую.

Несколько проще задача получается, когда заданы только две границы

$r(0,v)$ и $r(1,v)$. Линейная интерполяция в u-направлении позволяет получить ли-

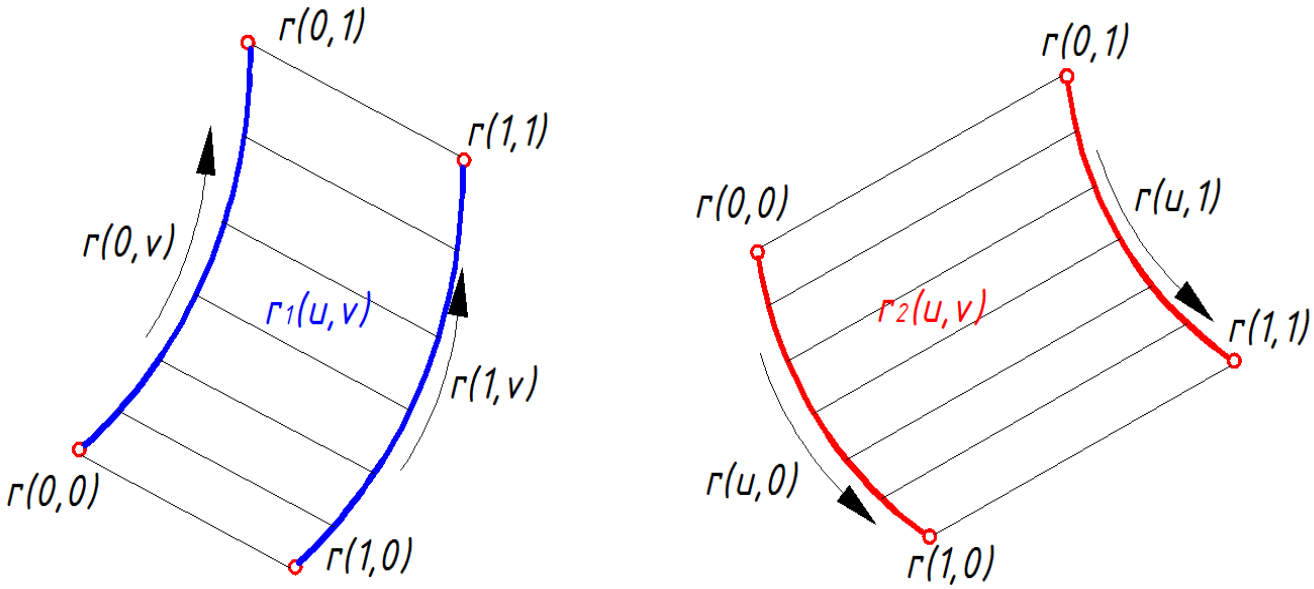


Рисунок 24 – Отсеки поверхностей Кунса

нейчатую поверхность:

$$r_1(u,v) = (1-u)r(0,v) + ur(1,v). \qquad (43)$$

Напротив, линейная интерполяция в v-направлении дает поверхность, удовлетворяющую двум другим граничным условиям (смотри рисунок 18)

$$r_2(u,v) = (1-v)r(u,0) + vr(u,1). \qquad (44)$$

Их сумма r_1+r_2 представляет отсек поверхности, каждая из границ которой, является суммой заданной граничной кривой и прямолинейного отрезка, соединяющего концевые точки этой кривой. Тогда можно найти отсек поверхности $r_3(u\ v)$, границами которой служат вышеупомянутые прямолинейные отрезки.

Его границы, отвечающие $v=0$ и $u=0$, будут определяться выражениями:

$$S_1(u,v) = (1-u)r(0,0) + ur(1,0) \text{ и } S_2(u,v) = (1-u)r(0,1) + ur(1,1). \qquad (45)$$

Последующая линейная интерполяция в v-направлении даст вектор $r_3(u,v)$

$$r_3(u,v)=(1-v)(1-u)r(0,0)+u(1-v)r(1,0)+v(1-u)r(0,1)+vr(1,1) \qquad (46)$$

который и определит искомую поверхность $r = r_1 + r_2 - r_3$.

Вспомогательные функции u, $(1-u)$, v и $(1-v)$ называются функциями смешения, так как они соединяют воедино четыре отдельные граничные

кривые, чтобы дать одну корректно определенную поверхность.

Полученная формула допускает возможность обобщения. Линейные функции смешения в уравнении отсека возникают в результате использования равномерной линейной интерполяции и могут быть заменены на $\alpha_1(u)$, $\alpha_2(u)$, где функции смешения теперь будут любые, для которых выполняются соотношения: $\alpha = 1-\alpha_1$ и $\alpha(0)=1$, $\alpha(1)=0$, $\alpha_1(0)=0$, $\alpha_1(1)=1$.

При такой замене интерполяция в *u*-направлении остается линейной, но скорость движения конца вектора *r* вдоль линии интерполяции теперь не остается постоянной при равномерном возрастании *u* от *0* до *1*. Соответствующая замена $(1-v), v$ на $\alpha(м)$, $\alpha_1(м)$ приводит к формуле

$$r(u,v)=[\alpha(u)\alpha_2(u)]R+[r(u,0)r(u,0)]A-[\alpha(u)\ \alpha_2(u)]SA, \qquad (47)$$

где $R=[r(u,0)r(u,1)]^m$, $A=[\alpha_o(v)\ \alpha_2(v)]^m$,

$$S = \begin{bmatrix} r(0,0) & r(0,1) \\ r(1,0) & r(1,1) \end{bmatrix}. \qquad (48)$$

Функции смешения α и α_1 выбираются обычно непрерывными и монотонными. На практике, как правило, используют полиномиальные функции смешения.

§ 7 Отсеки поверхностей в форме Bezier

Кривые в форме Безье полностью располагаются внутри выпуклой оболочки. Отсек поверхности Безье также может быть спроектирован с помощью выпуклой оболочки — характеристического многогранника, который определяется векторами положения r_{ij} его вершин.

Применительно к кубической форме отсек поверхности в некотором смысле аппроксимирует многогранник (смотри рисунок 25), хотя только точки r_{00}, r_{03}, r_{30} и r_{33} являются у них общими. Форма многогранника дает общее представление о соответствующем отсеке поверхности, и изменение одного или более векторов r_{ij} модифицирует ее предсказуемым образом.

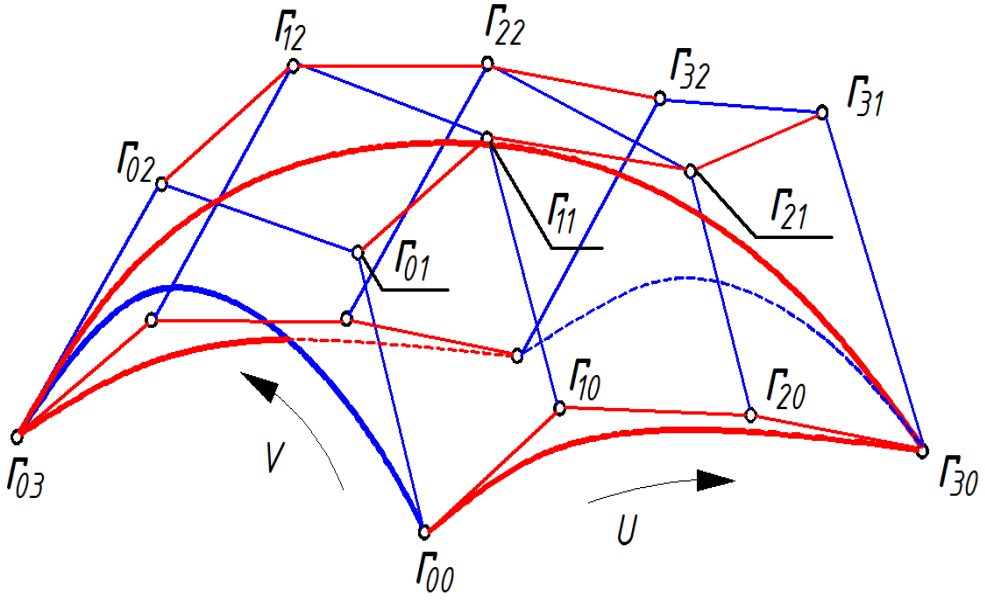


Рисунок 25 – Поверхность Bezier

Уравнение, определяющее каждый отсек поверхности, имеет вид

$$r(u,v) = \Sigma\Sigma\, r_{ij} g_i(u),\ g_j(v),\ 0 \le u, v \le 1, \qquad (49)$$

где $g_k(t) = 3![t^k(1-t)^{3-k}]/[k!(3-k)!]$, $k = 0,\ldots,3$.

Функции g_k суть кубические базисные полиномы Бернштейна. Конец вектора $r(0,v)$ описывает кубическую кривую Безье, характеристическая ломаная которой определяется векторами r_{00}, r_{03}, r_{30} и r_{33}. Очевидно, что эта кривая есть одна из границ порции. Три другие границы порции даются соответственно векторами $r(1,v)$, $r(u, 0)$, $r(u,1)$ и являются кривыми того же типа.

Из уравнения отсека поверхности можно получить значения векторов градиента и кручения в углах порции через векторы вершин характеристического многогранника. Векторы градиента зависят только от граничных вершин, в то время как четыре внутренние точки r_{11}, r_{12}, r_{22} и r_{21} влияют на величины частных производных.

Однако при проектировании отсека поверхности нет необходимости в задании векторов градиента или кручения. Достаточно знать векторы положения r_{ij} — 16 вершин многогранника.

Рассмотрим возможность обеспечения гладкости составной поверхности

Безье. Пусть имеются две смежные порции кубической поверхности Безье, заданные соответственно уравнениями:

$$r^{(1)}(u,v)=UMB^{(1)}M^{m}V \text{ и } r^{(2)}(u,v) = UMB^{(2)}M^{m}V. \quad (50)$$

Непрерывность составной поверхности на границе отсеков будет обеспечена, если $r^{(l)}(1,v) = r^{(2)}(0,v)$ при $v \in [0,1]$. С учетом этого условие непрерывности можно представить в виде:

$$[1\ 1\ 1\ l]UMB^{(1)}M^{m}V = [1\ 0\ 0\ 0]UMB^{(2)}M^{m}V. \quad (51)$$

Обе части этого уравнения являются кубическими полиномами от v. После преобразования это уравнение примет вид:

$$[1\ 1\ 1\ l]UMB^{(1)}M^{m} = [1\ 0\ 0\ 0]UMB^{(2)}M^{m}, \quad (52)$$

а после умножения справа на $(M^{m})^{-1}$ сведется к следующим четырем соотношениям: $r^{(l)}_{3i} = r^{(2)}_{0i}, i = 0,\ldots,3.$

Это означает выполнение вполне естественного условия: общая граничная кривая между двумя отсеками требует наличия общей граничной ломаной у двух характеристических многогранников.

Для непрерывности градиента при переходе через границу касательная плоскость к отсеку 1 при $u = l$ должна совпадать с касательной плоскостью к отсеку 2 при $u = 0$ для $v = [0,1]$. Тогда направление нормали к составной поверхности будет изменяться непрерывно при переходе через общую границу, и, следовательно, выполняется условие:

$$r_u^{(2)}(0,v) \times r_v^{(2)}(0,v) = \lambda(v) r_u^{(1)}(1,v) \times r_v^{(1)}(1,v). \quad (53)$$

Присутствие положительной скалярной функции $\lambda(v)$ учитывает разрывы модуля вектора нормали к поверхности.

В общем случае уравнение отсека поверхности может быть в виде:

$$r(u,v) = \Sigma\Sigma\, r_{ij}\, g_i^p(u),\, g_j^q(v). \quad (54)$$

где $g_i^p(u)$ и $g_j^q(v)$ Базисные функции Бернштейна степени p и q соответственно, а отсек задается характеристическим многогранником с $(p+q)\times(q+1)$ вершинами.

§ 8 Поверхности, определяемые B-сплайнами

B-сплайновые кривые, рассмотренные выше, связаны с поверхностями, определяемыми B-сплайнами, точно так же, как кривые Безье связаны с поверхностями Безье. Подобно отсеку поверхности Безье, отсек B-сплайновой поверхности задается характеристическим многогранником. Последний весьма хорошо имитирует общие геометрические свойства отсека, которые определяются для прямоугольного массива вершин многогранника r_{ij}, $i=0,\ldots,p$, $j=0,\ldots,q$ формулой:

$$r(u,v) = \Sigma\Sigma r_{ij} N_{4,i+1}(u) N_{4,j+1}(v). \qquad (55)$$

Здесь, для построения отсека, используются кубические B-сплайны,

$(0 \le u \le p-2,\ 0 \le v \le q-2)$. Свойства B-сплайнов гарантируют, что построенная поверхность будет повсюду иметь непрерывность градиента и кривизны.

B-сплайны являются только локально ненулевыми, изменение формы одного отсека (без изменения границ) не влечет изменения формы соседних отсеков. Последнее позволяет весьма эффективно проводить локальное редактирование составной поверхности.

Глава 2 Наглядные изображения и проецирование

Изображение пространственных объектов, независимо от того, где это происходит (на бумаге или экране дисплея) осуществляется при помощи их

проекций (или проекций элементов им принадлежащих). В процессе проецирования каждая точка предмета отображается на плоскости проекций (образ точки называют проекцией).

Если лини проекции параллельны, то имеет место параллельное проецирование. Если же линии проекции сходятся в одной общей точке P, то

получаемое изображение называется центральной проекцией, или перспективным изображением предмета.

§1 Параллельное проецирование

При параллельном проецировании точки объекта проецируются на плоскость проекции параллельно некоторому направлению u.

Точка A в плоскости проекции П может быть определена при помощи соответствующей плоской системы координат. Для фиксации плоскости проекции и системы координат в ней необходимо, используя исходную пространственную систему координат, ввести три вектора r_0, u_1 и u_2. Для этого совмещают

начало плоской системы координат и конец вектора r_0. Оси плоской системы совмещают (по направлению) с вектором u_1 и u_2 (смотри рисунок 26). Посколь-

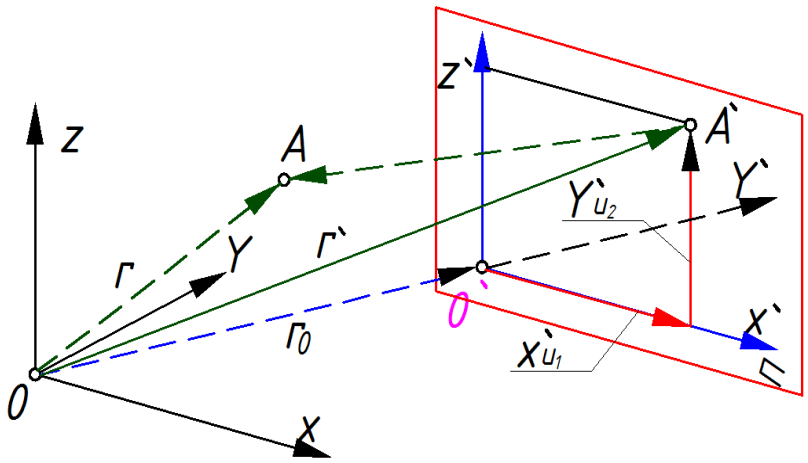


Рисунок 26 – Параллельное проецирование

ку каждая точка предмета r проецируется параллельно u в r', то $r'=r-z'u$.

Если x' и y' — координаты проекции произвольной точки, то

$$r'= r_0+x'u_1+y'u_2 \text{ и тогда } r'=r-z'u= r_0+x'u_1+y'u_2.$$

При этом координатные оси не обязательно ортогональны.

Беря же скалярное произведение последнего уравнения на $u_1' \; u_2$ и исключая тем самым x', y', мы получаем z'. Таким образом,

$$z` = \frac{(r - r_0)(u_1 \times u)}{u(u_2 \times u_1)}, \tag{56}$$

А скалярные же произведения этого уравнения на $u_1' \; u$ и $u_2' \; u$, дают соответственно x' и y':

$$\begin{cases} x' = \dfrac{(r - r_0)(u_1 \times u)}{u(u_2 \times u_1)}, \\ y' = \dfrac{(r - r_0)(u_2 \times u)}{u(u_2 \times u_1)} \end{cases} \tag{57}$$

В большинстве случаев плоскость проекции расположена перпендикулярно линиям проекции, т. е. $u=u_1' \; u_2$, и указанные выше уравнения имеют более простой вид : $x' = (r-r_0)u_1$, $y' = (r-r_0)u_2$, $z' = (r-r_0)u_3$.

Стандартная ортогональная аксонометрия может быть построена по упрощенным выражениям, полученным исходя из следующих соображений.

Углы между проекциями ортогональных осей однозначно определены. Построение аксонометрического изображения точки сводится к построению координатной ломаной x_a-$y_a z_a$ в соответствии с рисунком 27.

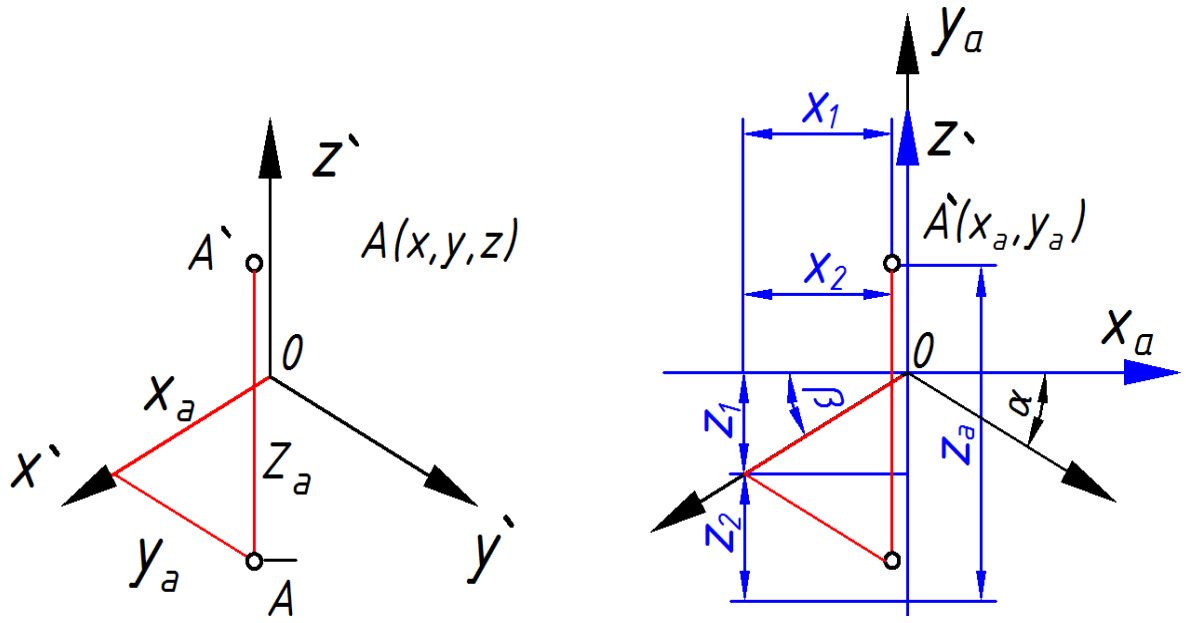


Рисунок 27 – Аксонометрическое изображение точки

Совмещение начала координат и проекции аксонометрической оси oZ’ с осью Декартовой системы координат на плоскости Ya позволяет записать следующее:

$$\begin{cases} Xa = - x_1+x_2, \\ Ya =-z_1-z_2+z_a \end{cases} \qquad (58)$$

С учетом коэффициентов искажения по осям k_x,k_y и k_z эти выражения примут вид:

$$\begin{cases} Xa =-xk_xCos\beta +yk_yCos\alpha, \\ Ya =-xk_xSin\beta -yk_ySin\alpha +z_ak_y \end{cases} \qquad (59)$$

Для построения стандартной приведенной изометрии, у которой $k_x=k_y=k_z$ и $\alpha =\beta=30°$ уравнения, преобразующие пространственные координаты объекта в координаты на плоскости, будут иметь вид:

$$\begin{cases} Xa =(y-x)Cos30° , \\ Ya =z-(y+x)Sin30° \end{cases} \qquad (60)$$

§ 2 Центральное проецирование (перспектива)

В линейной перспективе проекция r', прообраз r и точка r_v (точка наблюдения) коллинеарны в соответствии с рисунком 28.

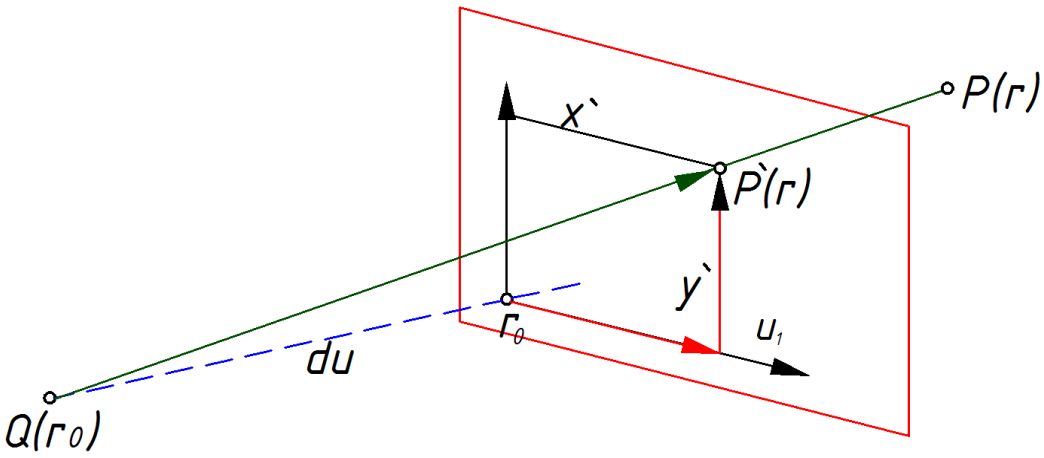


Рисунок 28 – Центральное проецирование

Пусть, как и раньше, r_0 – начало системы координат плоскости проекции, а u_1 и u_2, - векторы осей координат и $r' = z' \cdot r' + x' u_1 + y \cdot u_2$.

Вследствие того, что проекция точки лежит на прямой, соединяющей r и r_v, для некоторого значения z: $r` = z` \cdot r + (1 - z')Un.$

Таким образом: $r' = z' \cdot r' + x' u_1 + y' \cdot u_2 = z' r + (1 - z')Uv.$ Умножая скалярно это уравнение на векторы $u_2 \cdot (r - r_v)$, $u_1 \cdot (r - r_v)$ и $u_1 \cdot u_2$, получим соответственно значения координат x', y' и $z`$.

Если расстояние от точки обзора до плоскости проекции равно d, то $r_v = r_0 + du$, где $u = u_1 \cdot u_2$,. Отсюда:

$$\begin{cases} x` = -\dfrac{d(r - r_0)u_1}{(r - r_0)u - d}, \\ y` = -\dfrac{d(r - r_0)u_2}{(r - r_0)u - d}, \\ z` = -\dfrac{d}{(r - r_0)}u - d, \end{cases} \qquad (61)$$

Обычно начало координат перспективного изображения выбирается таким образом, чтобы прямая, соединяющая точку обзора с началом координат,
была перпендикулярна плоскости проекции.

Глава 3 Тени

Простой способ определения объектов, попавших в тень и, следовательно, неосвещенных, аналогичен алгоритму удаления невидимых поверхностей: те объекты, которые невидимы из источника освещения, но видимы из точки зрения - находятся в тени. На первом шаге в алгоритме с учетом тени определяются все многоугольники, видимые из точки освещения. Затем выполняется удаление поверхностей невидимых из точки зрения. При

выполнении закраски многоугольника проверяется, не закрыт ли он многоугольником, видимым из источника освещения. Если да, то в модели освещения учитываются (если надо) все три компоненты - диффузное и зеркальное отражения и рассеянный свет. Если же перекрытия нет, то закрашиваемый многоугольник находится в тени и надо учитывать только рассеянный свет.

Если положения наблюдателя и источника света совпадают, то теней не видно, но они появляются, когда наблюдатель перемещается в любую другую точку. Изображение с построенными тенями выглядит гораздо реалистичнее, и, кроме того, тени очень важны для моделирования. Например, особо интересующий нас участок может оказаться невидимым из-за того, что он попадает в тень. В прикладных областях - строительстве, разработке космических аппаратов и др. - тени влияют на расчет падающей солнечной энергии, обогрев и кондиционирование воздуха.

Наблюдения показывают, что тень состоит из двух частей: полутени и полной тени. Полная тень - это центральная, темная, резко очерченная часть, а полутень - окружающая ее более светлая часть. В машинной графике обычно рассматриваются точечные источники, создающие только полную тень. Распределенные источники света конечного размера создают как тень, так и полутень: в полной тени свет вообще отсутствует, а полутень освещается частью распределенного источника. Из-за больших вычислительных затрат, как правило, рассматривается только полная тень, образуемая точечным источником света. Сложность и, следовательно, стоимость вычислений зависят и от положения источника. Легче всего, когда источник находится в бесконечности, и тени определяются с помощью ортогонального проецирования. Сложнее, если источник расположен на конечном расстоянии, но вне поля зрения; здесь необходима перспективная проекция. Самый трудный случай, когда источник находится в поле зрения. Тогда надо делить пространство на секторы и искать тени отдельно для каждого сектора.

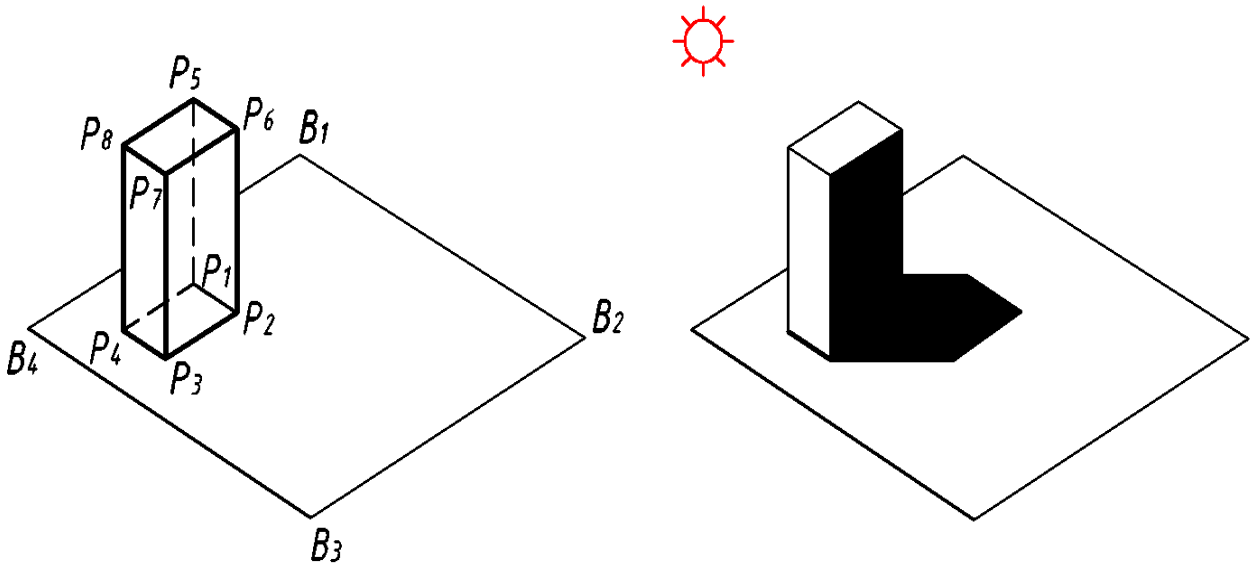
Для того чтобы построить тени, нужно по существу дважды удалить невидимые поверхности: для положения каждого источника и для положения наблюдателя или точки наблюдения, то есть это двухшаговый процесс. Рассмотрим сцену на рисунок 29. Один источник находится в бесконечности сверху: спереди слева от параллелепипеда. Точка наблюдения лежит спереди: сверху справа от объекта. В данном случае тени образуются двояко: это собственная тень и проекционная. Собственная тень получается тогда, когда сам объект препятствует пропаданию света на некоторые его грани, например на правую грань параллелепипеда. При этом алгоритм построения теней аналогичен алгоритму удаления нелицевых граней: грани, затененные собственной тенью, являются нелицевыми, если точку наблюдения совместить с источником света.

Если один объект препятствует попаданию света на другой, то получается проекционная тень, например тень на горизонтальной плоскости на рисунке 29.а. Чтобы найти такие тени, нужно построить проекции всех нелицевых граней на сцену. Центр проекции находится в источнике света. Точки пересечения проецируемой грани со всеми другими плоскостями образуют многоугольники,

которые помечаются как теневые многоугольники и заносятся в структуру дан ных. Для того чтобы не вносить в нее слишком много многоугольников, можно проецировать контур каждого объекта, а не отдельные грани.

После добавления теней к структуре данных, строится вид сцены из заданной точки наблюдения. Отметим, что для создания разных видов не нужно вычислять тени заново, так как они зависят только от положения источника и не зависят от положения наблюдателя.

Пример – Рассмотрим параллелепипед на рисунке 29.а. Он задан точками



а) б)

Рисунок 29 – Алгоритм построения теней

$P_1(1, 0, 3.5)$, $P_2(2, 0, 3.5)$, $P_3(2, 0, 5)$, $P_4(1, 0, 5)$, $P_5(1, 3, 3.5)$, $P_6(2, 3, 3.5)$, $P_7(2, 3, 5)$, $P_8(1, 3, 5)$. Параллелепипед стоит на плоскости, заданной точками $B_1(0, 0, 0)$, $B_2(6, 0, 0)$, $B_3(6, 0, 6)$, $B_4(0, 0, 6)$. Источник света расположен в бесконечности на прямой, проходящей через P_2 и P_8. Точка наблюдения находится в бесконечности на положительной полуоси z после поворота сцены сначала на -45° вокруг оси y и затем на 35° вокруг оси X.

Для того чтобы найти собственные тени, необходимо определить нелицевые грани относительно положения источника. Составим матрицу объема параллелепипеда, где *R, L, B, T, H, Y* обозначают правую, левую, нижнюю, верхнюю, ближнюю и дальнюю грани, если смотреть на преобразованный объект из бесконечности с положительной полуоси *Z*.

Вектор от источника к объекту в однородных координатах:

$$[E] = P_2 - P_8 = [1, -3, -1.5, 0].$$

Существует несколько способов нахождения проекционных теней. Один из них заключается в том, чтобы перенести и повернуть параллелепипед вместе с плоскостью его основания до совмещения вектора направления на источник с осью *Z*. Источник находится в бесконечности, поэтому ортогональная проекция видимых граней на преобразованную плоскость основания дает проекционную

грань. Значение Z получается подстановкой X и Y - координат вершин преобразованного параллелепипеда в уравнение преобразованной плоскости основания. Затем координаты проекционных теней проводятся к первоначальной ориентации.

Для совмещения вектора падающего из бесконечности вдоль прямой P_8P_2 света с осью X необходимо:

- перенести P_2 в начало координат;
- выполнить поворот на 33.69^o вокруг оси Y, чтобы P_4 совпала с осью Z;
- выполнить поворот на 59.04^o вокруг оси Y, чтобы P_8 совпала с осью Z.

Объединенное преобразование:

Уравнение преобразованной плоскости основания найдем методом Ньюэлла:

$$Z = -0.6\ Y.$$

Подставляя X и Y - координаты вершин преобразованного параллелепипеда в уравнение плоскости, определим Z, что дает проекцию тени на плоскость основания:

Штрих обозначает вершину проекционной тени.

Из положения источника видны только передняя, левая и верхняя грани; они и порождают проекционные тени:

передняя: $P_3P_4P_8P_7 - P_3'P_4'P_8'P_7'$

левая: $P_1P_4P_8P_5 - P_1'P_4'P_8'P_5'$

верхняя: $P_7P_8P_5P_6 - P_7'P_8'P_5'P_6'$

Отметим, что ни одна видимая грань не содержит точку P_2; поэтому ее проекция P_2' не входит в видимые проекционные тени. Обратным преобразованием, то есть $[T]^{-1}$, построенные проекционные тени переводятся в исходную ориентацию:

Проекциями теней на плоскость основания будут $S_3S_4S_8S_7$, $S_1S_4S_8S_5$, $S_7S_8S_5S_6$, $S_4S_8S_7$, $S_1S_4S_8S_5$, $S_7S_8S_5S_6$, а общим контуром - $S_1S_5S_6S_7S_3S_4$.

На рисунке 29.б показан результат после поворота на -45^o вокруг оси у и затем на 35^o вокруг оси X. Точка наблюдения находится в бесконечности на

положительной полуоси *Z*. Правая грань видима, но находится в собственной тени, поэтому она выглядит почти черной. Проекционная тень тоже выглядит почти черной, причем при наблюдении из заданной точки часть ее оказывается невидимой.

Идея совмещения процессов построения теней и удаления невидимых поверхностей была впервые предложена Аппелем. Им был разработан как метод трассировки лучей, так и метод построчного сканирования, усовершенствованный впоследствии Букнайтом и Кели. Включение теней в интервальный алгоритм построчного сканирования, например в алгоритм Уоткинса, осуществляется в два этапа.

На первом этапе для каждого многоугольника сцены и каждого источника определяются самозатененные участки и проекционные тени. Они записываются в виде двоичной матрицы, в которой строки - многоугольники, отбрасывающие тень, а столбцы - затеняемые многоугольники. Единица в матрице означает, что грань может отбрасывать тень на другую, ноль - что не может. Единица на диагонали соответствует многоугольнику в собственной тени.

Рассмотрим для иллюстрации несложный пример.

Обычно матрица включается в связный список, объединяющий тени и многоугольники.

Второй этап - обработка сцены относительно положения наблюдателя - состоит из двух процессов сканирования. В интервальном алгоритме построчного сканирования, например Уоткинса, первый процесс определяет, какие отрезки на интервале видимы. Второй с помощью списка теневых многоугольников находит, падает ли тень на многоугольник, который создает видимый отрезок на данном интервале. Второе сканирование для интервала производится следующим образом:

а) если нет ни одного теневого многоугольника, то видимый отрезок изображается;

б) если для многоугольника, содержащего видимый отрезок, имеются теневые многоугольники, но они не пересекают и не покрывают данный интервал, то видимый отрезок изображается;

в) если интервал полностью покрывается одним или несколькими теневыми многоугольниками, то интенсивность изображаемого видимого отрезка определяется с учетом интенсивностей этих многоугольников и самого отрезка;

г) если один или несколько теневых многоугольников частично покрывают интервал, то он разбивается в местах пересечения с ребрами теневых многоугольников. Затем алгоритм применяется рекурсивно к каждому из подинтервалов до тех пор, пока интервал не будет изображен.

Здесь предполагается, что интенсивность видимого отрезка зависит от интенсивности теневого многоугольника. В простейшем случае можно считать, что тени абсолютно черные. Однако, если взять источник света и два каких-нибудь предмета, то можно увидеть, что тень может быть не только такой. Интенсивность, то есть чернота тени, зависит от интенсивности источника и от расстояния между затененной гранью и гранью, отбрасывающей тень. Это вызвано ограниченным размером источника и тем, что на затененную поверхность падает рассеянный свет.

Для того чтобы смоделировать такой эффект, можно установить пропорциональную зависимость интенсивности тени и источника. Если накладывается несколько теней, то их интенсивности складываются. Более сложных расчетов требует правило, позволяющее сделать интенсивность тени пропорциональной как интенсивности источника, так и расстоянию между поверхностью, на которую падает тень, и поверхностью, отбрасывающей тень.

Можно изменить алгоритм, использующий Z-буфер, так, чтобы он включал построение теней. Модифицированный алгоритм состоит из двух шагов:

1) строится сцена из точки наблюдения, совпадающей с источником. Значения z для этого вида хранятся в отдельном теневом Z-буфере. Значения интенсивности не рассматриваются;

2) затем сцена строится из точки, в которой находится наблюдатель. При обработке каждой поверхности или многоугольника его глубина в каждом пикселе сравнивается с глубиной в Z-буфере наблюдателя. Если поверхность видима, то значения X, Y, Z из вида наблюдателя линейно преобразуются в значения X', Y', Z' на виде из источника. Для того чтобы проверить, видимо ли значение Z' из положения источника, оно сравнивается со значением теневого Z-буфера при X', Y'. Если оно видимо, то оно отображается в буфер кадра в точке X, Y без изменений. Если нет, то точка находится в тени и изображается согласно соответствующему правилу расчета интенсивности с учетом затенения, а значение в Z-буфере наблюдателя заменяется на Z'.

Для этого метода можно непосредственно использовать алгоритм построчного сканирования с Z-буфером. В этом случае применяется буфер размером с одну сканирующую строку. Уильямс модифицировал метод, чтобы строить криволинейные тени на изогнутых поверхностях. Сначала создается вид из точки наблюдения, а затем, как постпроцесс, выполняется поточечное линейное преобразование к виду из источника и построение теней. Уильямс отметил, что таким способом неправильно изображаются блики: в тени они просто становятся темнее, хотя ясно, что там их вообще не должно быть. Уильямсом также были исследованы эффекты квантования, возникающие в пространстве изображения в результате преобразования от одной точки наблюдения к другой.

Азертон включил построение теней в алгоритм удаления невидимых поверхностей, основанный на методе отсечения Вейлера-Азертона . Его преимущество состоит в том, что он работает в объектном пространстве и результаты годятся как для точных расчетов, так и для синтеза изображений. Процесс состоит из двух шагов.

На первом шаге с помощью алгоритма удаления невидимых поверхностей выделяются освещенные, то есть видимые из положения источника грани. Для повышения эффективности алгоритма в памяти хранятся именно они, а не грани, лежащие в тени. Если хранить теневые, то есть невидимые многоугольники, то придется запоминать и все не лицевые грани объекта, которые обычно отбрасываются до применения алгоритма удаления невидимых поверхностей. Для выпуклого многогранника это удвоило бы количество обрабатываемых граней.

Освещенные многоугольники помечаются и преобразуются к исходной ориентации, где они приписываются к своим прототипам в качестве многоугольников детализации поверхности. Эта операция выполняется путем присвоения своего номера каждому многоугольнику сцены. В процессе удаления невидимых граней многоугольник может быть разбит на части, которые сохраняют тот же номер. Поэтому каждый кусок освещенной грани можно связать с соответствующим исходным многоугольником или с любой его частью.

Для того чтобы не получить ложных теней, сцену надо рассматривать только в пределах видимого или отсекающего объема, определенного положением источника. Иначе область вне этого объема окажется затененной, и наблюдатель увидит ложные тени. Это ограничение требует также, чтобы источник не находился в пределах сцены, так как в этом случае не существует перспективного или аксонометрического преобразования с центром в источнике, которое охватывало бы всю сцену.

На втором шаге объединенные данные о многоугольниках обрабатываются из положения наблюдателя. Если какая-либо область не освещена, применяется соответствующее правило расчета интенсивности с учетом затенения.

Если источников несколько, то к базе данных добавляется несколько наборов освещенных граней.

Алгоритм выделения видимых поверхностей трассировкой лучей также можно расширить, чтобы включить построение теней. Процесс вновь делится на два этапа. На первом, как и в предыдущем случае, трассировкой луча от точки наблюдения через плоскость проекции определяются видимые точки сцены (если таковые есть).

На втором этапе вектор (луч) трассируется от видимой точки до источника света. Если между ними в сцене есть какой-нибудь объект, то свет от источника не попадает в данную точку, то есть она оказывается в тени (смотри рисунок 30).

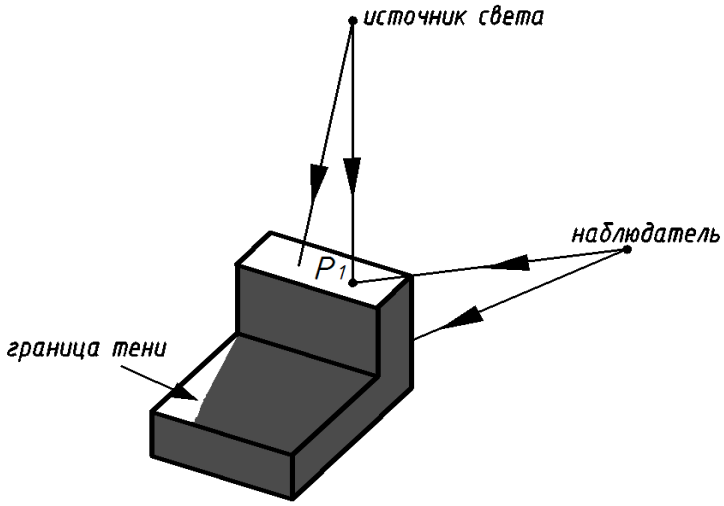


Рисунок 30 – Алгоритм выделения видимых поверхностей трассировкой лучей

Кук предложил довольно простой способ построения полутеней, хотя, как уже говорилось, обычно они не учитываются. В модели освещения Кука - Торрэнса источнику конечного размера противолежит телесный угол *dr*, поэтому, закрывая часть источника, можно уменьшить телесный угол, а следовательно, и интенсивность падающего от источника света. При этом соответственно снижается и отраженная интенсивность.

Это показано на рисунке 31 для прямого края четырехугольника и сферического источника. Средняя линия полутени рассчитывается в предположении, что точечный источник находится в центре сферы. С помощью

подобных треугольников найдем проекцию половины ширины полутени *r* на направление *L* (смотри рисунок 31): r(nL)/d = R/D где d- расстояние от точки, отбрасывающей тень, до соответствующей точки на средней линии полутени; *D* - расстояние от точки, отбрасывающей тень, до центра сферического источника; *R* - радиус сферического источника.

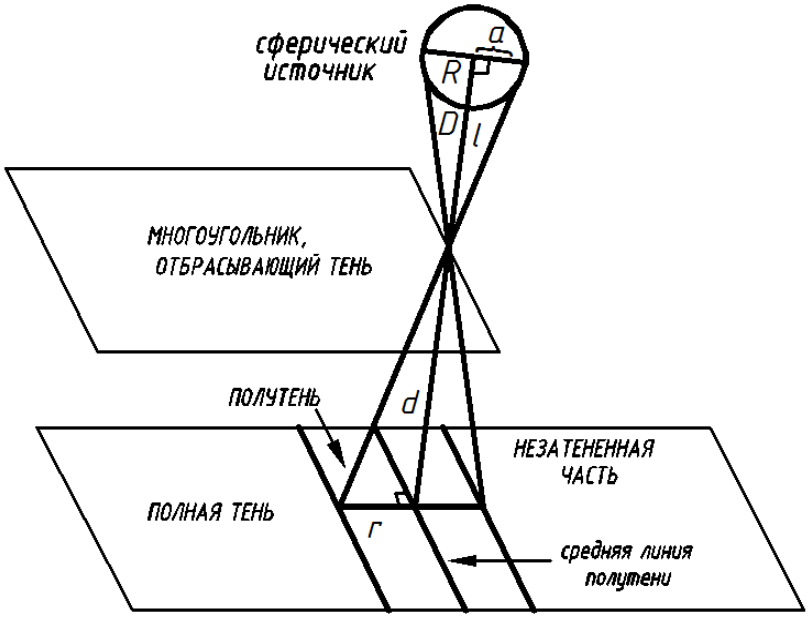


Рисунок 31 – Способ построения полутеней

Если смотреть от многоугольника, отбрасывающего тень, то телесный угол источника d есть $d = (R/D)^2$ поэтому половина ширины полутени равна

$$r = \frac{d}{(n'L')} \cdot \frac{R}{D} = \frac{d}{(n'L')} \cdot \left(\frac{d\omega}{\pi} \right)^{1/2}.$$

Это означает, что если телесный угол источника меньше, то он создает более резкую тень (то есть с меньшим *r*). У точечного источника $d\square = 0$ и $r = 0$, поэтому полутени нет вообще. При сближении затеняемой и затеняющей поверхности *d* и *r* уменьшаются, и тень становится резче.

Интенсивность точек полутени определяется видимой частью источника. Для сферического источника, частично видимого от - *R* до *a*, эта доля составляет:

$$A_{frac} = \frac{1}{2} + \frac{1}{\pi} \cdot \left[\frac{a}{R} \cdot \sqrt{1 - \left(\frac{a}{R} \right)^2} + Sin^{-1}\left(\frac{a}{R} \right) \right]$$

Результаты показывают, что на одном краю полутень получается более четкой. Кук рекомендует хранить эти данные в таблице цветов. Однако проще рассчитывать линейную аппроксимацию $A_{frac} = 0.5(1 + a/R)$, которая дает погрешность меньше 7 %.

Часть 4 Работа с векторными графическими системами (AutoCAD)

Успехи компьютерных технологий, достигнутые в последние годы, не оставляют места сомнениям при выборе способов получения, хранения и переработки данных о сложных комплексных трехмерных объектах, таких, например, как памятники архитектуры и археологии, объекты спелеологии и т. д. Несомненно, что применение компьютеризации для этих целей – дело не далекого будущего, а уже настоящего времени. Последнее, конечно, в большой мере зависит от количества денежных средств, вкладываемых с этой целью.

Системы CAD/CAM используются сегодня в различных областях инженерной конструкторской деятельности от проектирования микросхем до создания самолетов. Ведущие инженерные и производственные компании, такие как Boeing, в конечном счете, двигаются к полностью цифровому представлению конструкции самолетов.

Если в растровой графике базовым элементом изображения является точка, то в векторной графике - линия. Как и любой объект, линия обладает свойствами: формой (прямая, кривая), толщиной, цветом, начертанием (сплошная, пунктирная). Замкнутые линии приобретают свойство заполнения. Охватываемое ими пространство может быть заполнено другими объектами (текстуры, карты) или выбранным цветом. Линия описывается математически как единый объект, и потому объем данных для отображения объекта средствами векторной графики существенно меньше, чем в растровой графике.

Важно и то, что векторные изображения могут быть увеличены или уменьшены без потери качества. Это возможно, т.к. масштабирование

изображений производится с помощью простых математических операций (умножение параметров графических примитивов на коэффициент масштабирования). Векторные графические изображения являются оптимальным средством для хранения высокоточных графических объектов (чертежей, схем и т.д.), для которых имеет значение сохранение четких и ясных контуров. Среди большого числа автоматизированных систем формирования изображений, таких как CorelDraw, CorelHara, Designer и др. особняком стоит AutoCAD фирмы Autodesk. На сегодняшний день это, пожалуй, самая распространенная система. Первая версия AutoCADa была разработана в 1982 году. В 1989 году появилась первая русифицированная версия AutoCAD10. В настоящее время широко эксплуатируются версии AutoCAD-2008 и AutoCAD-2009. Создана Ассоциация пользователей этой графической системой.

AutoCAD – универсальная графическая система, в основу структуры которой положен принцип открытой архитектуры, позволяющий адаптировать и развивать многие функции AutoCAD применительно к конкретным задачам и требованиям.

Комплекс AutoCAD – известная программная продукция, предназначенная для автоматизации проектно – конструкторских работ. По своим возможностям он может быть отнесен к системам САПР среднего уровня. Такие системы, как правило, предоставляю возможность автоматизированного параметрического твердотельного моделирования трехмерных объектов, состоящих из нескольких сотен компонентов, обеспечивают организацию проектно – конструкторских работ подразделений предприятия, обмен данными с другими САПР, автоматизацию документооборота и выпуск конструкторской документации.

Скорость и легкость, с которыми создаются трехмерные модели проектируемых изделий, широкие возможности их преобразования и редактирования, различные способы получения плоских изображений этих изделий (видов, разрезов, сечений), ассоциативно связанных с моделями – все

это обеспечивает огромную экономию времени по сравнению с «ручным» черчением.

Современный пакет AutoCAD позволяет работать одновременно с несколькими чертежами, имеет мощные средства визуализации создаваемых трехмерных объектов и расширенные возможности адаптации системы к требованиям пользователя, обеспечивает связь графических объектов с внешними базами данных, позволяет просматривать и копировать компоненты чертежа без открытия его файла, редактировать внешние ссылки и блоки, находящиеся во внешних файлах.

Приложения к системе AutoCAD, разработанные различными фирмами, позволяют на основе трехмерной модели быстро определить прочностные характеристики проектируемого изделия методом конечных элементов и скорректировать геометрию модели, выполнить расчеты кинематики и динамики механизма, моделировать и исследовать его работу без изготовления дорогостоящей модели прототипа, осуществлять технологическую подготовку производства пресс-форм и разверток деталей, получаемых методом листовой штамповки и гибки, решать задачи для подготовки программ для 2-, 3-, и 4-координатных фрезерных и электроэрозионных станков.

САПР: автоматическая система черчения и моделирования.

AutoCAD работает с такими примитивами, как: точка; отрезок; круг; дуга; полилиния (последовательность прямолинейных и дуговых сегментов с возможным указанием ширины); эллипс; кольцо; многоугольник; сплайн (гладкая кривая, проходящая через заданные точки); фигура (закрашенные треугольники и четырёхугольники на плоскости); полоса (плоская закрашенная линия любой заданной ширины, одинаковой по всей длине); прямая; луч; текст.

Часть 5 Рекурсия и фракталы

Глава 1 Рекурсия и итерация

До сих пор мы рассматривали либо линейные программы, либо программы с ветвлениями, не требовавшими многократного повторения одних и тех операций. Гораздо чаще, однако, проходится иметь дело именно с такими ситуациями. Существует два основных подхода к решению задач подобного типа: рекурсия и итерация.

Определение 1 Рекурсия - это такой способ организации обработки данных, при котором, программа вызывает сама себя непосредственно, либо с помощью других программ.

Определение 2 Итерация - способ организации обработки данных, при котором определенные действия повторяются многократно, не приводя при этом к рекурсивным вызовам программ.

Математическая модель рекурсии заключается в вычислении рекурсивно определенной функции на множестве программных переменных. Примерами таких функций могут служить факториал числа и числа Фибоначчи. В каждом из этих случаев значение функции для всех значений аргумента, начиная с некоторого, определяется через предыдущие значения.

Факториал $n!$ целого неотрицательного числа n задается следующими соотношениями:

$$0! = 1,$$

$$n! = n \cdot (n-1)! \quad \text{для} \quad n > 0.$$

Числами Фибоначчи f_n называют последовательность величин 0, 1, 1, 2, 3, 5, 8, , … определяемую равенствами:

$$f_0 = 0,$$

$$f_1 = 1,$$

$$f_n = f_{n-1} + f_{n-2} \quad \text{для} \quad n > 1.$$

Математическая модель итерации сводится к повторению некоторого преобразования (отображения) $T: X \to X$ на множестве переменных программы X (прямом произведении множеств значений отдельных

переменных). Программной реализацией итерации является обычно некоторый цикл, тело которого осуществляет преобразование T.

В качестве примера можно рассмотреть схему вычисления факториала натурального числа в соответствии с его другим определением: $n! = 1 \cdot 2 \cdot \ldots n$

При написании программы в соответствии с ним нужно работать с двумя величинами целого типа Z_M: числом i которое будет играть роль счетчика и изменяться от 1 до n включительно, и величиной k, в которой будет накапливаться произведение чисел от 1 до i.

Пространством X в данном случае будет Z^2_M, в качестве начальной точки в этом пространстве возьмем точку (1,1) (что соответствует $i = k = 1$), а преобразование $T: X \to X$ будет иметь вид *T(i.k)=(i+1,k*i)* В случае, например, трехкратного применения преобразования T получим:

$$T(T(T(1,1))) = T(T(2,1)) = T(3,2) = 6,$$

что обеспечит вычисление факториала числа 3.

Рекурсия и итерация взаимозаменяемы. Более точно, справедливо следующее утверждение.

Теорема 1. Любой алгоритм, реализованный в рекурсивной форме, может быть переписан в итерационном виде, и наоборот.

Данная теорема не означает, что временная и емкостная эффективности получающихся программ обязаны совпадать.

Рекурсия - это такой способ организации обработки данных, при котором программа вызывает сама себя непосредственно, либо с помощью других программ.

Итерация - способ организации обработки данных, при котором определенные действия повторяются многократно, не приводя при этом к рекурсивным вызовам программ.

Любой алгоритм, реализованный в рекурсивной форме, может быть переписан в итерационном виде, и наоборот.

Мы говорим, что функция рекурсивна (или, что она основана на рекурсии), если в ней содержится одно или несколько обращений к самой себе

или к другим функциям, в которых есть обращения к данной функции. При входе в обычную функцию выход из нее всегда происходит раньше, чем повторный вход, но для рекурсивной функции это необязательно.

Глава 2 Графика и случайные числа

Иногда нам не нужна полная симметрия, возникающая при прямом применении рекурсии. В таких случаях можно использовать (псевдо-) случайные числа для устранения в некоторой степени такой симметрии. Этим способом с помощью программы была получена картинка на рисунок 32.

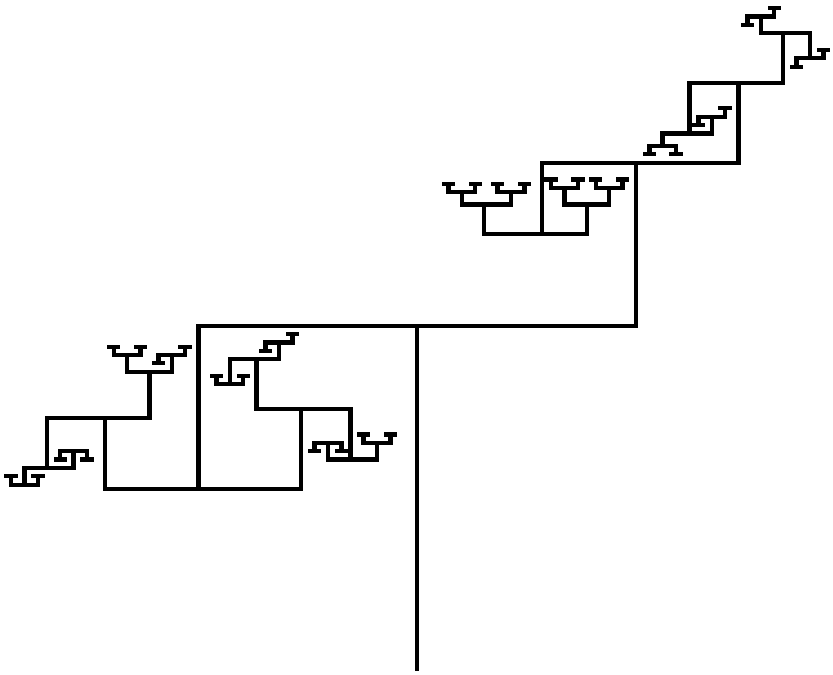


Рисунок 32 – Рекурсия на основе случайных чисел

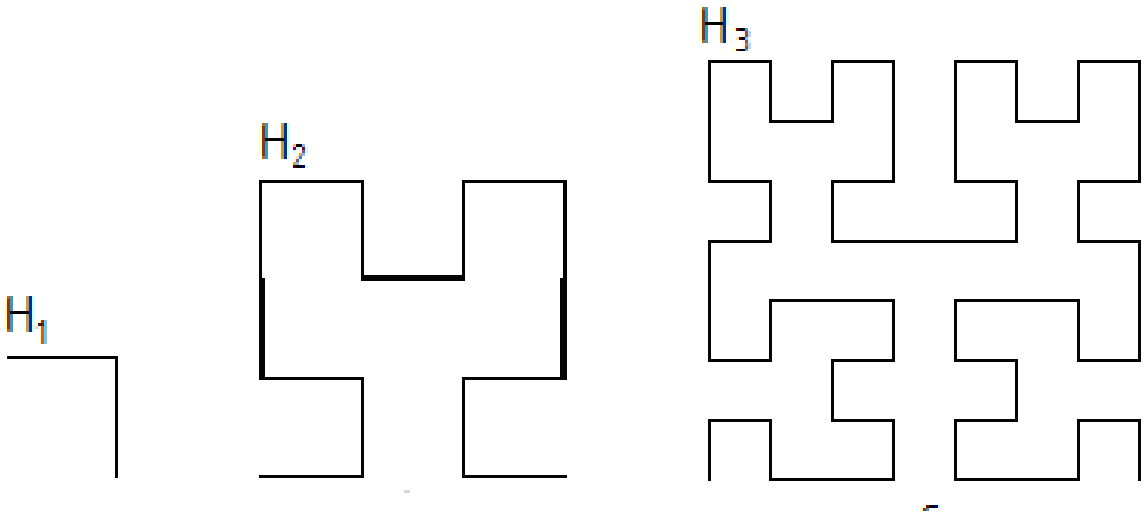
Здесь показано многократное изображение большой буквы Т. Точку в нижней части буквы Т назовем ее начальной точкой. Начнем с большой буквы Т в ее нормальном положении, а концевые точки на верхней горизонтальной полке будут в свою очередь начальными точками новых букв Т, несколько меньших, чем первоначальная. От пользователя программы запрашивается ввод

двух коэффициентов уменьшения (fx и fy), глубины рекурсии и порогового значения в процентах. После вычерчивания каждой буквы Т генерируется случайное число, меньше 100. Если это число меньше, чем заданное пороговое значение, то новая буква Т вычерчивается в нормальном положении, в противном случае - в перевернутом. В программе используется хорошо известная формула $1 + r + r^2 + ... + r^{n-1} = (1 - r^n)/(1 - r)$ для вычисления размера первой буквы Т, такого, чтобы окончательный результат разместился в пределах границ экрана.

Глава 3 Кривые Гильберта

Рекурсия может быть использована для получения линейного рисунка, известного под названием *кривая Гильберта*. Кривая Гильберта основана на изображении буквы П, вычерченной в виде трех сторон квадрата, как показано на рисунке 33.а. Существуют кривые Гильберта порядков 1, 2, ..., обозначаемые как H_1, H_2,... На рисунке 33.б изображена кривая порядка H_2, в которой некоторые отрезки прямых линий вычерчены в виде толстых линий. Это так называемые *связки* (в действительности связки должны иметь одинаковую толщину с другими отрезками, здесь же они показаны утолщенными единственно с целью демонстрации способа получения H_2 из H_1).

Видно, что H_2 можно рассматривать как большую букву П, четыре части которой заменены меньшими по размеру буквами П. Эти меньшие буквы П соединены тремя связками. Каждая сторона меньшей буквы П имеет ту же длину, что и связка, они в три раза меньше стороны квадрата, в который вписывается



а) б) в)

Рисунок 33 – Кривые Гильберта 1, 2, 3порядков

H_2. Применим ту же процедуру к каждой из четырех букв П, составляющих H_2, то есть каждую букву П в H_2 заменим меньшей H_2, одновременно уменьшим длину связок так, чтобы их длины стали равными длине элементарного отрезка прямой линии, которые содержатся в трех малых фигурах H_2. Таким образом мы получим фигуру H_3, показанную на рисунке 33.в.

Теперь все элементарные отрезки в семь раз меньше, чем длина сторон квадрата, в который вписывается фигура H_3. Отсюда получаем, что коэффициенты уменьшения для этих элементарных отрезков в фигурах H_1, H_2, H_3, ... образуют ряд чисел: 1, 3, 7, ..., то есть в общем случае коэффициент уменьшения для фигуры H_n может быть вычислен по формуле: $2^n - 1$(смотри рисунок 34).

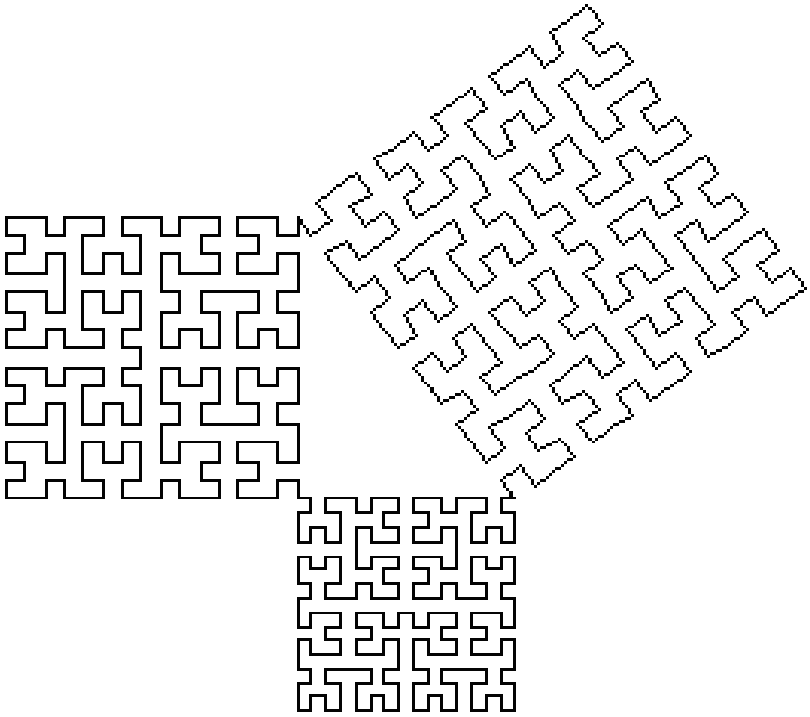


Рисунок 34 – Кривая Гильберта

Заметим, что связки в фигуре H_2 вычерчиваются в тех же направлениях, как и три отрезка, образующие букву П в фигуре H_1. При желании эти последние отрезки прямых линий можно рассматривать как связки, соединяющие четыре точки, которые в свою очередь можно принять за кривую Гильберта нулевого порядка.

Глава 4 Фракталы и фрактальная графика

Существуют фигуры, описать которые невозможно с помощью векторной и растровой графики. Примером являются лист папоротника, облака, морозные узоры, береговая линия, клякса.

Если увеличивать изображения, выполненные в растровой или векторной графике, то постепенно получим прямую линию или точку. Увеличивая же эти фигуры и узоры, мы увидим, что они повторяются.

Такие графические изображения можно представить с помощью фрактальной графики. В отличие от растровой и векторной графики – фрактальная графика основана на чистом программировании. Причем изображения строятся на основе рекурсии, то есть на способности функции обращаться к самой себе или к другим функциям, в которых есть обращение к такой функции (смотри рисунок 35).

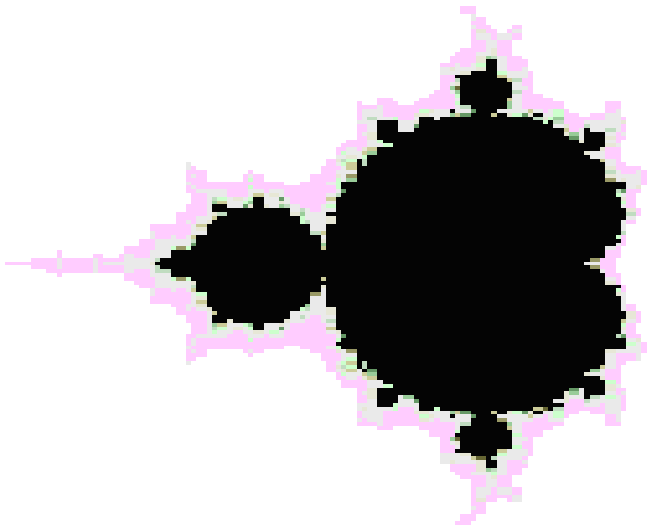


Рисунок 35 – Множество Мандельброта

Это самый что ни на есть обычный фрактал, построенный на основе множества Мандельброта. Подобные работы стали возможны благодаря созданию в 1970-х годах фрактальной геометрии, отцом которой заслуженно считается профессор Гарвардского университета Бенуа Мандельброт. Именно Мандельброт и придумал термин "фрактал" (от латинского *fractus* - дробленый). Фрак-

тал - это фрагментированная геометрическая форма, которая не обладает свойством непрерывности. Если взять какой-либо фрагмент такой структуры, он будет походить на уменьшенную копию исходного целого. Фракталы, как правило, самоподобны, их необычные свойства не зависят от масштаба. Фракталы окружают нас повсюду: падающая снежинка, причудливый морозный узор на стекле, сверкающая молния, прожилки опавшего листа.

Фракталы - это множества дробной размерности (не одномерные, не двумерные); это нечто промежуточное между точкой и линией, линией и поверхностью, поверхностью и телом. Множество Мандельброта (один и примеров фракталов) на плоскости математически определяется очень просто - это множество всех таких комплексных "c", при которых итерационная последовательность $z \rightarrow z^2 + c$ (начиная с $z = 0, c = x + iy$) остается ограниченной, не уходя на бесконечность.

Фракталы - это не просто математические абстракции, которые в силу своей красоты и изящества завораживают многих людей, причем даже тех, кто весьма далек от науки. Они дают качественно новое представление об окружающей нас природе. Это очень мощный инструмент для изучения хаоса. И в них, как в капельке росы, отражается наш реальный и придуманный мир - мир самоподобия.

Основное свойство фракталов – *самоподобие*. Любой микроскопический фрагмент фрактала в том или ином отношении воспроизводит его глобальную структуру. В простейшем случае часть фрактала представляет собой просто уменьшенный целый фрактал.

Файлы фрактальных изображений имеют расширение *fif*. Обычно файлы в формате *fif* получаются несколько меньше файлов в формате *jpg*, но бывает и наоборот. Самое интересное начинается, если рассматривать картинки со все большим увеличением. Файлы в формате *jpg* почти сразу демонстрируют свою дискретную природу – появляется пресловутая лесенка. А вот *fif* файлы, как и положено фракталам, с ростом увеличения показывают все новую степень детализации структуры, сохраняя эстетику изображения.

§1 Понятие размерности и ее расчет

В своей повседневной жизни мы постоянно встречаемся с размерностями. Мы прикидываем длину дороги, узнаем площадь квартиры и т.д. Это понятие вполне интуитивно ясно и, казалось бы, не требует разъяснения. Линия имеет размерность 1. Это означает, что, выбрав точку отсчета, мы можем любую точку на этой линии определить с помощью 1 числа – положительного или

отрицательного. Причем это касается всех линий – окружность, квадрат, парабола и т.д. (2^1).

Размерность 2 означает, что любую точку мы можем однозначно определить двумя числами. Не надо думать, что двумерный – значит плоский. Поверхность сферы тоже двумерна (ее можно определить с помощью двух значений – углов наподобие ширины и долготы).

Если смотреть с математической точки зрения, то размерность определяется следующим образом: для одномерных объектов – увеличение в два раза их линейного размера приводит к увеличению размеров (в данном случае длины) в два раза (2^2).

Для двумерных объектов увеличение в два раза линейных размеров приводит к увеличению размера (например, площадь прямоугольника) в четыре раза.

Для 3–х мерных объектов увеличение линейных размеров в два раза приводит к увеличению объема в восемь раз (2^3) и так далее.

Таким образом, размерность *D* можно рассчитать исходя из зависимости увеличения «размера» объекта *S* от увеличения линейных размеров *L*. *D=log(S)/ log(L)*. Для линии *D=log(2)/log(2)=1*. Для плоскости *D=log(4)/log(2)=2*. Для объема *D=log(8)/log(2)=3*.

Когда размерность фигуры получаемой из каких–то простейших объектов (отрезков) больше размерности этих объектов – мы имеем дело с фракталом.

На рисунке 36 представлена замкнутая кривая, которая называется *фрактальной кривой* или просто *фракталом*. Фрактал подобен острову с береговой линией, кажущейся ровной издали, но становящейся совершенно нерегулярной по мере приближения. Данную фигуру, можно построить с помощью рекурсии, причем с увеличением глубины рекурсии мы получим замкнутую область, граница которой становится очень большой по сравнению с самой областью.

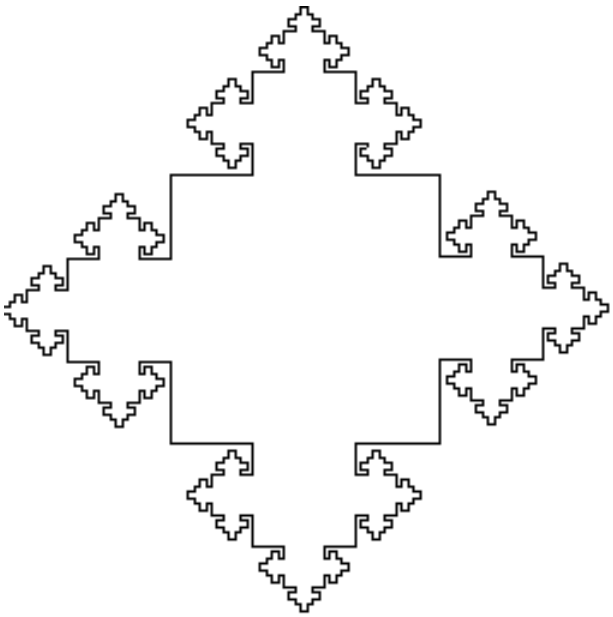


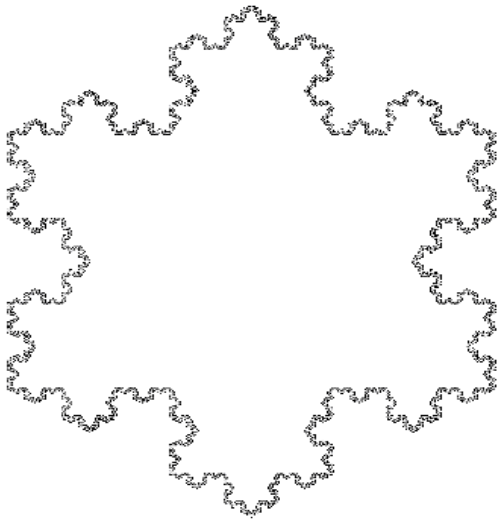
Рисунок 36 – Фрактал Гилберта

§ 2 Геометрические фракталы

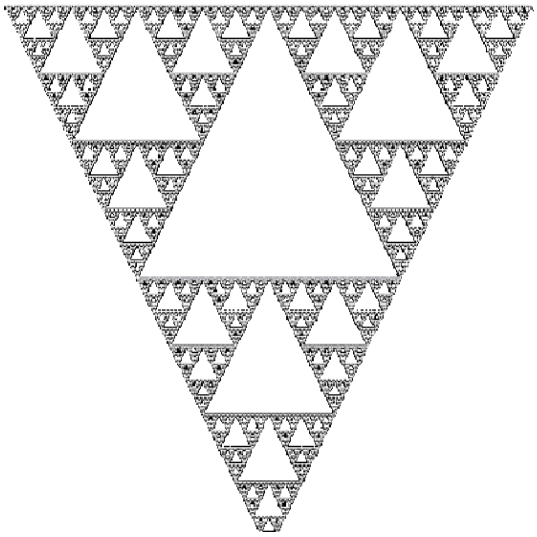
Именно с них и начиналась история фракталов. Этот тип фракталов получается путем простых геометрических построений. Обычно при построении этих фракталов поступают так: берется «затравка» – аксиома – набор отрезков, на основании которых будет строиться фрактал. Далее к этой «затравке» применяют набор правил, который преобразует ее в какую–либо геометрическую фигуру. Далее к каждой части этой фигуры применяют опять тот же набор правил. С каждым шагом фигура будет становиться все сложнее и сложнее, и если мы проведем бесконечное количество преобразований – получим *геометрический фрактал* (смотри рисунок 37).

Из этих геометрических фракталов очень интересным и довольно знаменитым является – *снежинка фон Коха (Коши- англ.)*. Строится она на основе равностороннего треугольника. Каждая линия которого заменяется на 4 линии каждая длинной в 1/3 исходной. Таким образом, с каждой итерацией длина кривой увеличивается на треть. И если мы сделаем бесконечное число

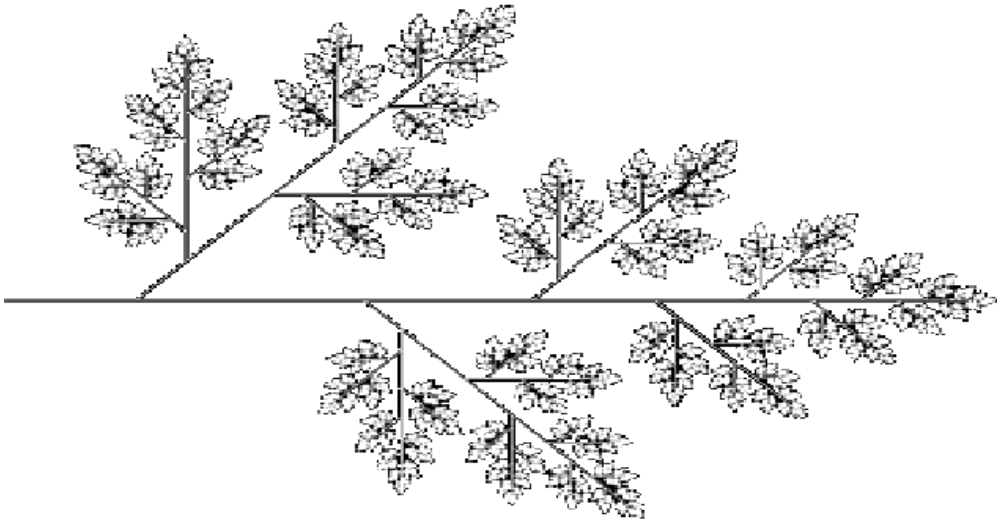
итераций – получим фрактал снежинку фон Коха бесконечной длины. Получается, что



а) Снежинка Коха



б) Треугольник Серпинского



в) Лист

Рисунок 37 – Геометрические фракталы

наша бесконечная кривая покрывает ограниченную площадь. Размерность снежинки фон Коха (при увеличении снежинки в 3 раза ее длина возрастает в 4 раза) D=log(4)/log(3)=1.2619...

Для построения геометрических фракталов хорошо приспособлены так называемые *L–Systems*. Суть этих систем состоит в том, что имеется определенный набор символов системы, каждый из которых обозначает определенное

действие и набор правил преобразования символов.

§ 3 Алгебраические фракталы

Вторая большая группа фракталов – *алгебраические*. Свое название они получили за то, что их строят, на основе алгебраических формул иногда весьма простых. Методов получения алгебраических фракталов несколько. Один из методов представляет собой многократный (итерационный) расчет функции *Zn+1=f(Zn),* где *Z* – комплексное число, а *f* некая функция. Расчет данной функции продолжается до выполнения определенного условия. И когда это условие выполнится – на экран выводится точка. При этом значения функции для разных точек комплексной плоскости может иметь разное поведение:

1) с течением времени стремится к бесконечности;

2) стремится к 0;

3) принимает несколько фиксированных значений и не выходит за их пределы;

4) поведение хаотично, без каких либо тенденций.

Чтобы проиллюстрировать алгебраические фракталы обратимся к классике – *множеству Мандельброта* (смотри рисунок 38).

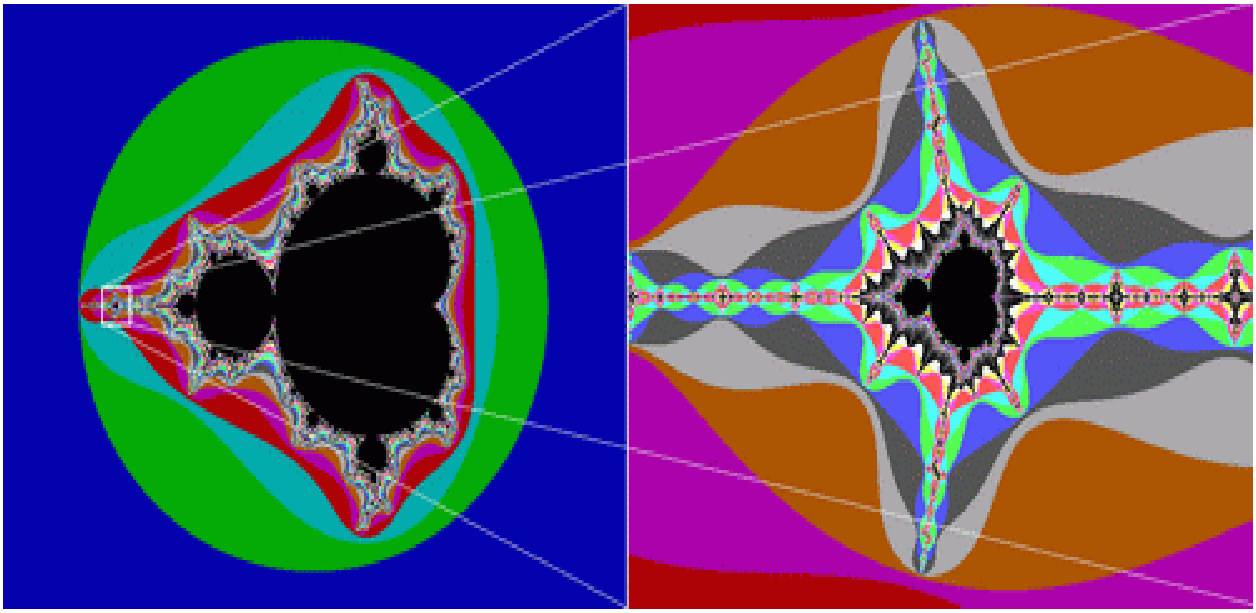


Рисунок 38 – Множество Мандельброта

Для его построения нам необходимы комплексные числа. Комплексное число – это число, состоящее из двух частей – действительной и мнимой, и обозначается оно *a+bi*. Действительная часть *a* - обычное число в нашем представлении, а *bi* – мнимая часть, *i* – называют мнимой единицей, потому, что если

мы возведем *i* в квадрат, то получим –1.

Комплексные числа можно складывать, вычитать, умножать, делить, возводить в степень и извлекать корень, нельзя только их сравнивать. Комплексное число можно изобразить как точку на плоскости, у которой координата *X* это действительная часть *a*, а *Y* это коэффициент при мнимой части *b*.

Функционально множество Мандельброта определяется как $Z_{n+1} = Z_n^2 + C$. Для всех точек на комплексной плоскости в интервале от *–2+2i* до *2+2i* выполняется некоторое достаточно большое количество раз $Z_n = Z_0^2 + C,$ каждый раз проверяя абсолютное значение Z_n. Если это значение больше 2, что рисуется точка с цветом равным номеру итерации на котором абсолютное значение превысило 2, иначе - точка черного цвета.

Черный цвет в середине показывает, что в этих точках функция стремится к нулю – это и есть *множество Мандельброта*. За пределами этого множества функция стремится к бесконечности. А самое интересное это границы множества. Они то и являются фрактальными. На границах этого множества функция ведет себя непредсказуемо – хаотично.

Меняя функцию, условия выхода из цикла можно получать другие фракталы.

§ 4 Стохастические фракталы

Типичный представитель данного класса фракталов «*Плазма*».

Для ее построения берется прямоугольник и для каждого его угла определяется цвет. Далее находится центральная точка прямоугольника и задается ее в цвет равный среднему арифметическому цветов по углам прямоугольника плюс некоторое случайное число. Чем больше случайное число – тем более «рваным» будет рисунок. Если, например, сказать, что цвет точки это высота над уровнем моря, то получим вместо плазмы – горный массив (смотри рисунок 39).

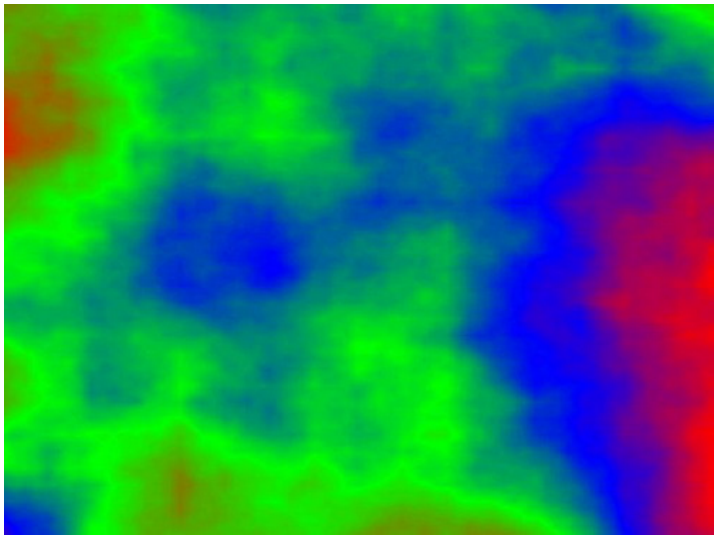


Рисунок 39 - Плазма

Именно на этом принципе моделируются горы в - большинстве программ. С помощью алгоритма, похожего на плазму строится карта высот, к ней применяются различные фильтры, накладывается текстура (смотри рисунок 40).

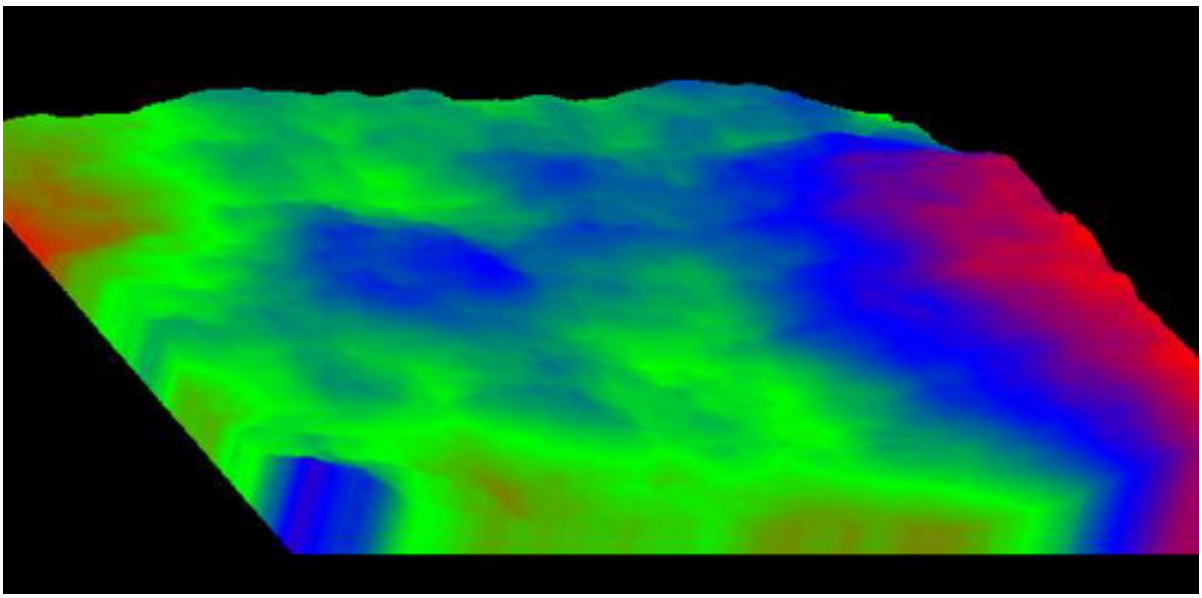
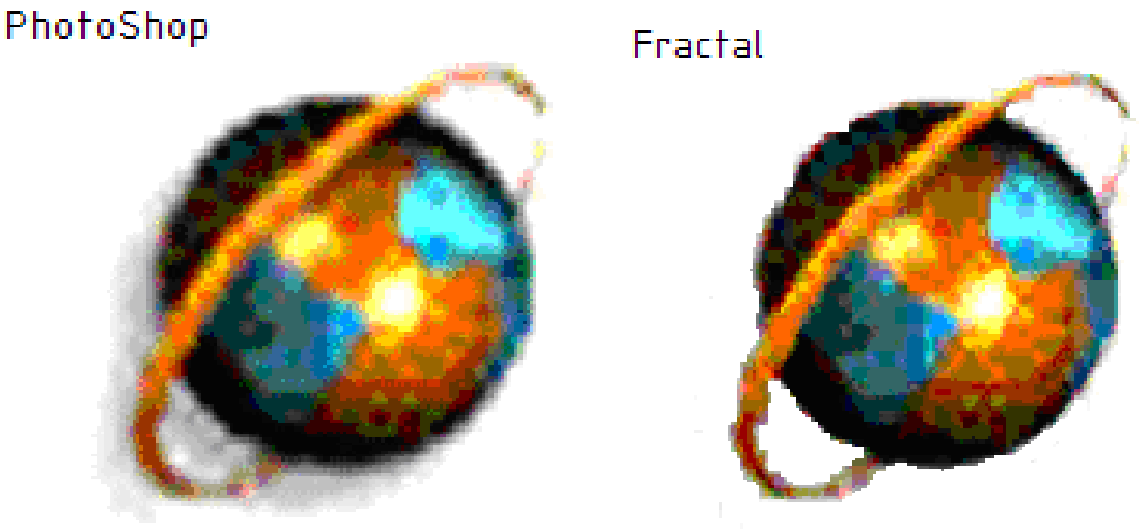


Рисунок 40 – Плазма 3D

§ 5 Системы итерируемых функций (IFS – Iterated Function Systems)

Эта группа фракталов получила широкое распространение благодаря работам Майкла Барнсли из технологического института штата Джорджия. Он пытался кодировать изображения с помощью фракталов. Запатентовав несколько идей по кодированию изображений с помощью фракталов, он основал фирму «Iterated Systems», которая через некоторое время выпустила первый продукт «Images Incorporated», в котором можно было изображения переводить из растровой формы во фрактальную FIF.

Это позволяло добиться высоких степеней сжатия. При низких степенях сжатия качество рисунков уступало качеству формата JPEG, но при высоких картинки получались более качественными. В любом случае этот формат не прижился, но работы по его усовершенствованию ведутся до сих пор. Ведь этот формат не зависит от разрешения изображения. Так как изображение закодировано с помощью формул, то его можно увеличить до любых размеров и при этом будут появляться новые детали, а не просто увеличится размер пикселей (смотри рисунок 41).



а) Рисунок формата JPEG б) IFS изображение

Рисунок 41 – Сравнение изображений

Если в L–systems (алгебраических фракталах) речь шла о замене прямой линии неким полигоном, то в IFS в ходе каждой итерации заменяется некий полигон (квадрат, треугольник, круг) на набор полигонов, каждый их которых подвергнут аффинным преобразованиям. При аффинных преобразованиях исходное изображение меняет масштаб, параллельно переносится вдоль каждой из осей и вращается на некоторый угол.

Часть 6 Теоретические сведения о метафайлах

Метафайл – файл, который содержит либо определяет другие файлы. Приставка «мета» означает более высокую абстракцию либо место в иерархии. Понятие метафайла, именуемого также файлом файлов, введено потому, что быстро расширяется область, связанная с обработкой данных и особенно обработкой изображений. А последняя, часто не кончается одним сеансом, и изображения нужно хранить с возможностью повторного использования. Их необходимо также передавать по сети из одних систем в другие.

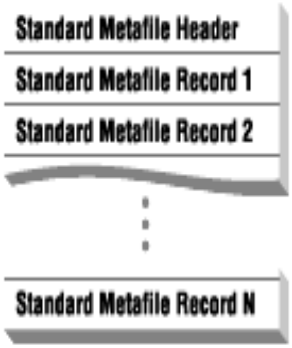
В этой связи возникла необходимость, а Международная Организация Стандартов (МОС) совместно с Международной Электротехнической Комиссией (МЭК) разработала и утвердила стандарт, именуемый Метафайлом компьютерной графики CGM. Стандарт определяет структуру метафайла, т.е. виды и места тех или иных его элементов. В метафайле предусмотрены три способа кодирования информации: литерное, двоичное и текстовое. Литерное кодирование осуществляется 7–8-битовыми числами. Двоичное кодирование облегчает выполнение процессов интерпретации данных. Текстовое кодирование требует большей памяти, но представляет данные в виде, удобном для пользователей. Документ CGM определяет метафайлы в ряде широко распространенных стандартах. К ним, в первую очередь, относятся сетевая служба ODA, функциональный профиль MAP, функциональный профиль TOP.

При создании документов необходима манипуляция с данными, состоящими из текста и изображений. Для этого ISO предлагает стандартный обобщенный язык разметки. В этом стандарте определены пять типов документов: доклад, отчет, подотчет, письмо, набор слайдов. Проведенная стандартизация позволяет хранить в системах графическую информацию в единообразном виде, обеспечивать передачу изображений между системами различных типов и разных производителей и предоставлять возможности интерпретации и редактирования графических данных.

Метафайл состоит из одного или двух информационных заголовков и массива переменной длины, который содержит информацию о реальных GDI командах.

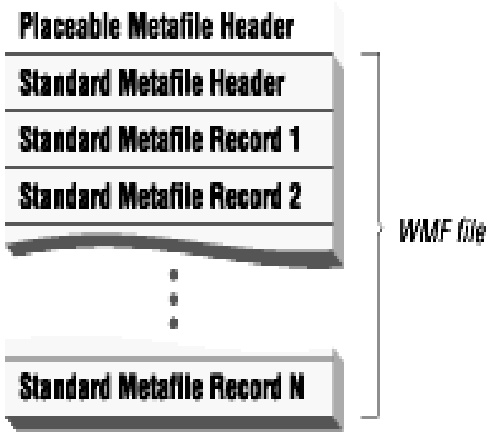
Есть четыре разновидности метафайлов *standard*, *placeable*, *clipboard*, и *enhanced*.

Метафайлы **standard** содержат 18-байтовый заголовок WMF, после которого следуют одна или более записей GDI команд. Последняя запись в файле содержит информацию, указывающую, что достигнут конец данных.



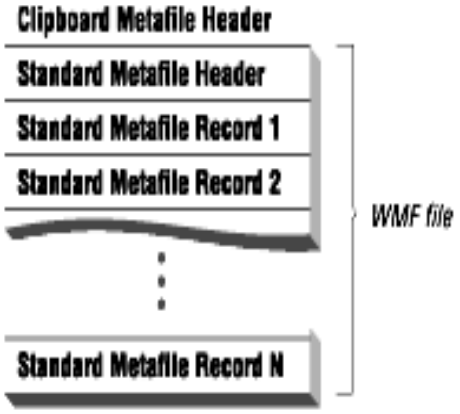
Microsoft Windows Metafile-1: Standard metafile format

Метафайлы **placeable** содержат дополнительный 22-байтовый заголовок, который содержит информацию, описывающую положение изображения на устройстве отображения. После него следует стандартный 18-байтовый заголовок WMF и командные записи GDI.



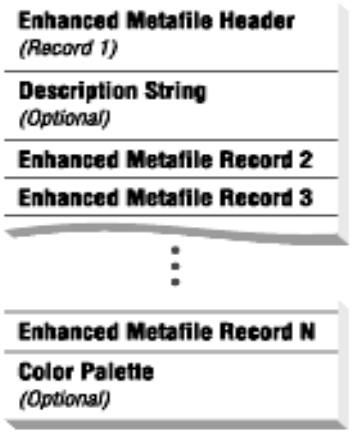
Microsoft Windows Metafile-2: Placeable metafile format

Метафайлы **clipboard** содержат дополнительный 8-байтовый (Win16) или 16-байтовый заголовок (Win32), который предшествует стандартному заголовку метафайла и описывает режим отображения и размер изображения.



Microsoft Windows Metafile-3: Clipboard metafile format

Enhanced метафайлы, как и предыдущие, содержат записи GDI команд, но с другим заголовком, являющимся объединением заголовков *placeable* и *clipboard* метафайлов. EMF заголовок также содержит строку описания файла и программируемую палитру.



Microsoft Windows Metafile-4: Enhanced metafile format

Стандартный заголовок Windows metafile занимает 18 байт и его стуктура выглядит следующим образом:

```
typedef struct _WindowsMetaHeader
{
  WORD  FileType;       /* Расположение метафайла (0=память, 1=диск) */
```

```
  WORD  HeaderSize;      /* Размер заголовка в словах (WORDS) (всегда =
9) */
  WORD  Version;         /* Версия Microsoft Windows */
  DWORD FileSize;        /* Общий размер метафайла в словах */
  WORD  NumOfObjects;    /* Количество объектов в файле */
  DWORD MaxRecordSize;   /* Размер самой большой записи в словах */
  WORD  NumOfParams;     /* Не используется (всегда 0) */
} WMFHEAD;
```

FileType содержит флаг, указывающий местоположение данных метафайла. *FileType*=0 указывает, что метафайл находится в памяти,*FileType*=1 указывает, что метафайл находится на диске.

HeaderSize содержит размер заголовка метафайла в словах, и всегда равно 9.

Version указывает версию Microsoft Windows создавшую этот метафайл.

Это значение должно быть всегда указано. Например, в метафайле созданном Windows 3.0 and 3.1, *Version*=0x0300.

FileSize определяет общий размер файла в словах.

NumOfObjects определяет количество объектов в файле.

MaxRecordSize определяет максимально возможный размер объекта, содержащегося в файле.

NumOfParams не используется. Всегда имеет значение 0.

Placeable метафайлы были разработаны корпорацией Aldus, как нестандартный путь указать параметры отображения на устройстве отображения. Placeable метафайлы широко распространены, но не поддерживаются Windows API. Для воспроизведения placeable метафайла используя Windows API необходимо переделать его заголовок. Каждый Placeable заголовок содержит 22 байта и выглядит следующим образом:

```
typedef struct _PlaceableMetaHeader
{
  DWORD Key;          /* Ключик (всегда 9AC6CDD7h) */
  WORD  Handle;       /* (всегда 0) */
  SHORT Left;         /* Левая координата */
  SHORT Top;          /* Верхняя координата */
  SHORT Right;        /* Правая координата */
  SHORT Bottom;       /* Нижняя координата */
  WORD  Inch;         /* Кол-во логических единиц в дюйме */
  DWORD Reserved;     /* Не используется (всегда 0) */
  WORD  Checksum;     /* Контрольная сумма для предыдущих 10 слов */
} PLACEABLEMETAHEADER;
```

Key содержит специальное значение, идентифицируцие заголовок и всегда равное 9AC6CDD7h.

Handle исспользуется при расположении метафайла в памяти. Когда файл находится на диске, всегда равен 0.

Left, Top, Right, and Bottom содержит координаты изображения на устройстве отображения в логических единицах. (1 логическая единица 1/1440 дюйма, 720 логических единиц = 1/2 дюйма).

Inch содержит количество логических единиц в дюйме для данного изображения. Обычно это 1440 логических единиц в дюйме.

Reserved не используется и всегда = 0.

Checksum содержит контрольную сумму предыдущих 10 слов заголовка. Вычисляется посредством XOR кодирования:

```
PLACEABLEMETAHEADER pmh;
pmh.Checksum = 0;
pmh.Checksum ^= (pmh.Key & 0x0000FFFFUL);
pmh.Checksum ^= ((pmh.Key & 0xFFFF0000UL) >> 16);
pmh.Checksum ^= pmh.Handle;
```

```
pmh.Checksum ^= pmh.Left;
pmh.Checksum ^= pmh.Top;
pmh.Checksum ^= pmh.Right;
pmh.Checksum ^= pmh.Bottom;
pmh.Checksum ^= pmh.Inch;
pmh.Checksum ^= (pmh.Reserved & 0x0000FFFFUL);
pmh.Checksum ^= ((pmh.Reserved & 0xFFFF0000UL) >> 16);
```

или: PLACEABLEMETAHEADER *pmh; WORD *ptr; pmh->Checksum = 0; for (ptr = (WORD *) pmh; ptr < (WORD *)pmh->Checksum; ptr++) pmh->Checksum ^= *ptr;

Clipboard метафайлы используют формат заголовка стандартных метафайлов, но ему предшествует 8- или 16-байтный заголовок:

```
typedef struct _Clipboard16MetaHeader
{
  SHORT MappingMode; /* Units used to playback metafile */
  SHORT Width;       /* Width of the metafile */
  SHORT Height;      /* Height of the metafile */
  WORD  Handle;      /* Handle to the metafile in memory */
} CLIPBOARD16METAHEADER;

typedef struct _Clipboard32MetaHeader
{
  LONG  MappingMode; /* Единицы, использующиеся для отображения */
  LONG  Width;       /* Ширина */
  LONG  Height;      /* Высота */
  DWORD Handle;      /* Указатель на область памяти */
} CLIPBOARD32METAHEADER;
```

MappingMode указывает тип координат для отображения. Может принимать следующие значения (смотри таблица 1):

Таблица 1

Значение	Mapping Mode	Одна логическая единица
1	Text	Один пиксел
2	Low Metric	0.1 миллиметра
3	High Metric	0.01 миллиметра
4	Low English	0.01 дюйм
5	High English	0.001 дюйм
6	Twips	1/1440 дюйма
7	Isotropic	Определяется приложением (aspect ratio preserved)
8	Anisotropic	Определяется приложением (aspect ratio not preserved)

Width и *Height* размер изображения с учетом значения *MappingMode*

Enhanced метафайлы являются новой и расширенной 32-битной версией стандартных метафайлов. Только 32-разрядный Windows API (Win32) поддерживает EMF файлы.

Строковые данные в EMF файлах используют Unicode кодировку.

Заголовок EMF содержит 80 байт и выглядит так:

```
typedef struct _EnhancedMetaHeader
{
   DWORD RecordType;       /* Тип записи */
   DWORD RecordSize;       /* Размер записи в байтах */
   LONG  BoundsLeft;       /* Левая граница */
   LONG  BoundsRight;      /* Правая граница */
   LONG  BoundsTop;        /* Верхняя граница */
   LONG  BoundsBottom;     /* Нижняя граница */
   LONG  FrameLeft;        /* Left side of inclusive picture frame */
   LONG  FrameRight;       /* Right side of inclusive picture frame */
   LONG  FrameTop;         /* Top side of inclusive picture frame */
```

```
    LONG  FrameBottom;      /* Bottom side of inclusive picture frame */
    DWORD Signature;        /* ID (Всегда 0x464D4520) */
    DWORD Version;          /* Версия метафайла */
    DWORD Size;             /* Размер файла  в байтах */
    DWORD NumOfRecords;     /* Количество записей в файле */
    WORD  NumOfHandles;     /* Number of handles in the handle table */
    WORD  Reserved;         /* Не используется (Всегда 0) */
    DWORD SizeOfDescrip;    /* Размер строки описания в словах */
    DWORD OffsOfDescrip;    /* Смещение строки описания */
    DWORD NumPalEntries;    /* Количество цветов в палитре */
    LONG  WidthDevPixels;   /* Ширина устройства в пикселях */
    LONG  HeightDevPixels;  /* Высота устройства в пикселях */
    LONG  WidthDevMM;       /* Ширина устройства в миллиметрах */
    LONG  HeightDevMM;      /* Высота устройства в миллиметрах */
        //Если версия Windows >= 4 то есть еще поля
    DWORD   cbPixelFormat;      // размер PIXELFORMATDESCRIPTOR
                        //Равен 0 если структура отсутствует
    DWORD   offPixelFormat;     // смещение PIXELFORMATDESCRIPTOR
                        // Равен 0 если структура отсутствует
     DWORD    bOpenGL;                // 1 если OpenGL присутствуют      //
команды OpenGL
} ENHANCEDMETAHEADER;
```

RecordType идентифицирует EMF запись. Для заголовка всегда 00000001h.

RecordSize размер заголовка в байтах.

BoundsLeft, BoundsRight, BoundsTop, и BoundsBottom указывает размер изображения в координатах X, Y, ширина, и высота. *BoundsTop* и *BoundsBottom* должны быть больше чем *BoundsLeft* и *BoundsRight* соответственно.

FrameLeft, FrameRight, FrameTop, и FrameBottom указывает размер страницы включающей изображение в координатах X, Y, ширина, и высота.*FrameTop* и *FrameBottom* должны быть больше чем*FrameLeft* и *FrameRight* соответственно.

Signature идентификатор. Всегда равен 0x464D4520.

Version версия EMF файла. Версия 1.0 выглядит как 0x00000100.

Size размер файла в байтах.

NumOfRecords количество записей метафайла, влючая заголовочную запись.

NumOfHandles количество заголовков, используемых метафайлом при размещении его в памяти. Всегда равен 0 для дисковых файлов.

Reserved не используется. Всегда 0.

SizeOfDescrip количество 16-битных Unicode символов строки описания, включая все NULL символы. Если параметр равен 0, то строки описания в файле нет.

OffsOfDescrip положение строки описания от начала файла. Если параметр равен 0, то строки описания в файле нет.

NumPalEntries количество цветов в палитре. Палитра, если она присутствует должна находиться в конечной записи файла. Если парамет равен 0, палитра отсутствует.

WidthDevPixels и *HeightDevPixels* ширина и высота устройства отображения в пикселах.

WidthDevMM и *HeightDevMM* ширина и высота устройства отображения в миллиметрах.

Строка описания EMF файла представляет собой набор Unicode символов. Размер строки практически не ограничен (может содержать более 2 билионов символов). Формат строки является «double NULL-terminated» где каждый первый 0 является признаком конца подстроки. Типичная строка описания выглядит следующим образом:

"Pain Paint 1.5\0Eating at Polly's\0\0"

После заголовка всех типов метафайлов следуют записи описания данных. Структура записей METARECORD описана в файле *WINDOWS.H* и выглядит следующим образом:

```
typedef struct _StandardMetaRecord
{
   DWORD Size;          /* общий размер записи в словах */
   WORD  Function;      /* Номер функции */
   WORD  Parameters[];  /* Параметры функции */
} WMFRECORD;
```

Size общий размер записи в словах, включая поля*Size* и *Function*. Минимальное значение для этого поля равно 3.

Function номер GDI функции вызываемой для воспроизведения записи. Младший байт указывает номер функции, старший – количество параметров функции в словах. Например, значение 0x0213 указывает на функцию LineTo() (0x13) и на то, что функции необходимо два параметра.

Параметры, используемые функцией, следуют в обратном порядке. Например, функции LineTo() необходимы параметры X и Y, но в записи они следуют, как Y и X.

Все параметры при сохранении метафайла приведены к типу WORD, Поэтому необходимо преобразовать их тип перед использованием в функции.

Последняя запись метафайлов всегда содержит функцию с номером 0000h, поле *Size* равно 0003h, отсутствует массив параметров. Эта запись является заглушкой и показывает, что достигнут конец данных. (Q99334)

Например, запись, описывающая функцию LineTo должна отобразить линию от текущей точки до точки, описанной в записи:

```
Size          5       /* 5 слов в записи */
Function      0x0213   /* номер функции LineTo */
```

```
Parameters[0]  50       /* координата Y */
Parameters[1]  100      /* координата X */
```

Эти данные приведут к вызову GDI функции LineTo():

```
LineTo(hDC, 100, 5);
```

Некоторые функции содержат не только массив параметров, но и структуры с данными. Например, запись, описывающая функцию BitBlt использует сохраненное растровое изображениев формате Device Dependent Bitmap (DDB). Структура записи BitBlt в метафайлах Windows 2.*x* выглядит следующим образом:

```
typedef struct _BitBltRecord
{
   DWORD     Size;            /* Размер записи в словах */
   WORD      Function;        /* Номер функции (0x0922) */
   WORD      RasterOp;        /* старшее слово растровой операции */
   WORD      YSrcOrigin;      /* Y-координата источника */
   WORD      XSrcOrigin;      /* X-координата источника */
   WORD      YDest;           /* ширина отображения */
   WORD      XDest;           /* высота отображения */
   WORD      YDestOrigin;     /* Y-координата отображения */
   WORD      XDestOrigin;     /* X-координата отображения */
   /* DDB Bitmap */
   DWORD     Width;           /* ширина растра в пикселях */
   DWORD     Height;          /* высота растра в строках */
   DWORD     BytesPerLine;    /* разрешение */
   WORD      NumColorPlanes;  /* количество цветов в палитре */
   WORD      BitsPerPixel;    /* количество бит на пиксель */
       RGBTRIPLE Bitmap[];        /* растр */
} BITBLTRECORD;
```

Массив *bitmap* состоит из структур RGBTRIPLE:

```
typedef struct _RGBTriple
{
   BYTE Red;
   BYTE Green;
   BYTE Blue;
} RGBTRIPLE;
```

DDB формат поддерживается Windows 2.*x* но не совместим с Windows 3.0 и старше. Windows 3.0 описывает функцию DibBitBlt и Device Independent Bitmap (DIB):

```
typedef struct _DibBitBltRecord
{
DWORD    Size;          /* Размер записи в словах */
   WORD     Function;       /* Номер функции (0x0922) */
   WORD     RasterOp;       /* старшее слово растровой операции */
   WORD     YSrcOrigin;     /* Y-координата источника */
   WORD     XSrcOrigin;     /* X-координата источника */
   WORD     YDest;          /* ширина отображения */
   WORD     XDest;          /* высота отображения */
   WORD     YDestOrigin;    /* Y-координата отображения */
   WORD     XDestOrigin;    /* X-координата отображения */
   /* DIB Bitmap */
       DWORD    Width;          /* ширина растра в пикселях */
   DWORD    Height;         /* высота растра в строках */
   DWORD    BytesPerLine;   /* разрешение */
   WORD     NumColorPlanes;  /* количество цветов в палитре */
```

```
   WORD      BitsPerPixel;     /* количество бит на пиксель */
   DWORD   Compression;      /* Формат сжатия */
   DWORD   SizeImage;        /* Размер растра в байтах */
   LONG    XPelsPerMeter;    /* расрешение по горизонтали */
   LONG    YPelsPerMeter;    /* разрешение по вертикали */
   DWORD   ClrUsed;          /* Количетсво используемых  цветов */
   DWORD   ClrImportant;     /* Количество важных цветов */
   RGBQUAD Bitmap[];         /* растр */
} DIBBITBLTRECORD;
```

Массив *bitmap*состоит из структур RGBQUAD:

```
typedef struct _RGBQuad
{
   BYTE Red;
   BYTE Green;
   BYTE Blue;
   BYTE Reserved;
} RGBQUAD;
```

Другая специфическая запись метафайлов, заслуживающая внимание это Escape (0x0626). API Windows 3.*x* поддерживает 64 escape-последовательностей, которые могут содержаться в метафайле. Но на практике каждая escape-последовательность работает по разному, в зависимости от типа принтера.

Ниже приведен список GDI функций, поддержка которых в метафайлах была изменена с Microsoft Windows 3.0:

AnimatePalette
BitBlt
CreatePalette

CreatePatternBrush

DeleteObject

DibBitBlt

DibCreatePatternBrush

DibStretchBlt

RealizePalette

ResizePalette

StretchBlt

Не все GDI функции могут быть использованы в метафайлах, а только те, которые используют заголовок устройства отображения в качестве первого параметра.

Enhanced метафайлы имеют свою собственную структуру, которая похожа на структуру записей стандартных метафайлов, но является полностью 32-разрядной:

```
typedef struct _EnhancedMetaRecord
{
   DWORD Function;      /* Номер функции  (определено в WINGDI.H) */
   DWORD Size;          /* Общий размер записи в байтах */
   DWORD Parameters[];   /* параметры */
} EMFRECORD;
```

Фунции, поддерживаемые EMFописаны в *WINDOWS.H* и начинаются с преффикса EMR_*.

Size общий размер записи в байтах, включая поля *Size* и *Function*. Минимальное значение равно 8.

Parameters массив параметров, используемых GDI функцией. Параметры следуют в обратном порядке, чем в вызове функции. Например, два параметра для LineTo(X, Y) будут следовать, как Y и X.

Таблица 2

Запись	Номер	Запись	Номер
1	2	3	4
EMR_ABORTPATH	68	EMR_POLYLINE	4
EMR_ANGLEARC	41	EMR_POLYLINE16	87
EMR_ARC	45	EMR_POLYLINETO	6
EMR_ARCTO	55	EMR_POLYLINETO16	89
EMR_BEGINPATH	59	EMR_POLYPOLYGON	8

Продолжение таблицы 2

1	2	3	4
EMR_BITBLT	76	EMR_POLYPOLYGON16	91
EMR_CHORD	46	EMR_POLYPOLYLINE	7
EMR_CLOSEFIGURE	61	EMR_POLYPOLYLINE16	90
EMR_CREATEBRUSHINDIRECT	39	EMR_POLYTEXTOUTA	96
EMR_CREATEDIBPATTERNBRUSHPT	94	EMR_POLYTEXTOUTW	97
EMR_CREATEMONOBRUSH	93	EMR_REALIZEPALETTE	52
EMR_CREATEPALETTE	49	EMR_RECTANGLE	43
EMR_CREATEPEN	38	EMR_RESIZEPALETTE	51
EMR_DELETEOBJECT	40	EMR_RESTOREDC	34
EMR_ELLIPSE	42	EMR_ROUNDRECT	44
EMR_ENDPATH	60	EMR_SAVEDC	33
EMR_EOF	14	EMR_SCALEVIEWPORTEXTEX	31
EMR_EXCLUDECLIPRECT	29	EMR_SCALEWINDOWEXTEX	32
EMR_EXTCREATEFONTINDIRECTW	82	EMR_SELECTCLIPPATH	67
EMR_EXTCREATEPEN	95	EMR_SELECTOBJECT	37
EMR_EXTFLOODFILL	53	EMR_SELECTPALETTE	48
EMR_EXTSELECTCLIPRGN	75	EMR_SETARCDIRECTION	57
EMR_EXTTEXTOUTA	83	EMR_SETBKCOLOR	25
EMR_EXTTEXTOUTW	84	EMR_SETBKMODE	18
EMR_FILLPATH	62	EMR_SETBRUSHORGEX	13
EMR_FILLRGN	71	EMR_SETCOLORADJUSTMENT	23
EMR_FLATTENPATH	65	EMR_SETDIBITSTODEVICE	80
EMR_FRAMERGN	72	EMR_SETMAPMODE	17
EMR_GDICOMMENT	70	EMR_SETMAPPERFLAGS	16
EMR_HEADER	1	EMR_SETMETARGN	28
EMR_INTERSECTCLIPRECT	30	EMR_SETMITERLIMIT	58
EMR_INVERTRGN	73	EMR_SETPALETTEENTRIES	50
EMR_LINETO	54	EMR_SETPIXELV	15
EMR_MASKBLT	78	EMR_SETPOLYFILLMODE	19
EMR_MODIFYWORLDTRANSFORM	36	EMR_SETROP2	20
EMR_MOVETOEX	27	EMR_SETSTRETCHBLTMODE	21
EMR_OFFSETCLIPRGN	26	EMR_SETTEXTALIGN	22

Продолжение таблицы 2

1	2	3	4
EMR_PAINTRGN	74	EMR_SETTEXTCOLOR	24

EMR_PIE	47	EMR_SETVIEWPORTEXTEX	11
EMR_PLGBLT	79	EMR_SETVIEWPORTORGEX	12
EMR_POLYBEZIER	2	EMR_SETWINDOWEXTEX	9
EMR_POLYBEZIER16	85	EMR_SETWINDOWORGEX	10
EMR_POLYBEZIERTO	5	EMR_SETWORLDTRANSFORM	35
EMR_POLYBEZIERTO16	88	EMR_STRETCHBLT	77
EMR_POLYDRAW	56	EMR_STRETCHDIBITS	81
EMR_POLYDRAW16	92	EMR_STROKEANDFILLPATH	63
EMR_POLYGON	3	EMR_STROKEPATH	64
EMR_POLYGON16	86	EMR_WIDENPATH	66

EMF файл также может содержать таблицу цветов, используемую при отображении изображения. По умолчанию, файл не содержит таблицу цветов и использует системную палитру.

Поле *NumPalEntries* заголовка EMF файла указывает на количество цветов в таблице и равно 0 если таблица отсутствует. Если таблица присутствует, то она находится в конечной записи-заглушке:

```
typedef struct _EndOfRecord
{
   DWORD  Function;       /* Номер функции (14) */
   DWORD  Size;           /* Размер записи в байтах */
   DWORD  NumPalEntries;  /* Количество  цветов палитре */
   DWORD  OffPalEntries;  /* Смещение первого  цвета */
   PALENT Palette[];      /* Палитра */
   DWORD  OffToEOF;       /* Смещение  от начала записи */
} ENDOFRECORD;
```

Function Номер функции. Для заглушки всегда равен 14.

Size общий размер записи в байтах. Для записи не содержаещй палитры равен 20.

NumPalEntries количество цветов в палитре. Значение переписывается из поля *NumPalEntries* заголовка.

OffPalEntries указывает позицию первого значения палитры от начала записи.

Palette массив стурктур PALENT. Поле отсутвует, если палитры нет.

OffToEOF смещение от начала записи. Равно значению поля *Size*.

Структура PALENT:

```
typedef struct _PaletteEntry
{
   BYTE Red;      /* */
   BYTE Green;    /* */
   BYTE Blue;     /* */
   BYTE Flags;    /* */
} PALENT;
```

Red, *Green*, и *Blue* содержат цветовые компоненты определения цвета.

Flags может принимать значения:

0x01	цвет используется для анимации
0x02	
0x04	Цвет не соответствует системной палитре

Одной из важных особенностей EMF формата является возможность использования скрытых данных в файле. В отличие от escape-последовательностей WMF, комментарий GDI может содержать любой тип данных, полностью независящий от внешних устройств

GDICOMMENT запись имеет формат:

```
typedef struct _GdiCommentRecord
{
   DWORD   Function;     /* Номер функции (70) */
   DWORD   Size;         /* Размер записи в байтах */
   DWORD   SizeOfData;   /* Размер данных в байтах */
   BYTE    Data[];       /* Комментарий */
} GDICOMMENTRECORD;
```

Function номер функции. Для GDI комментария всегда равен 70.

Size Общий размер записи в байтах. Для пустой записи комментария равен 12

SizeOfData Размер записи *Data* в байтах.

Data сам комментарий.

GDI комментарии могут содержать несколько определенных типов комментариев (смотри таблицу 3):

Таблица 3

Тип комментария	Содержимое
Public	Содержит метафайл
BeginGroup/EndGroup	Содержит коллекцию метафайлов и строк описания
Multiformats	Содержит метафайл и Encapsulated PostScript данные

Заголовок **Public** комментария выглядит следующим образом:

```
typedef struct _GdiCommentMetafile
{
   DWORD Identifier;      /* ID комментария(0x43494447) */
   DWORD Comment;         /* ID типа комментария (0x80000001) */
   DWORD Version;         /* Версия метафайла */
   DWORD Checksum;        /* Контрольная сумма метафайла */
   DWORD Flags;           /* Флаг (0) */
   DWORD Size;            /* Размер данных в байтах */
} GDICOMMENTMETAFILE;
```

Identifier содержит значение 0x43494447 идентифицирующее комментарий.

Comment содержит значение 0x80000001 идентифицирующее структуру **Public** комментария

Version содержит значение версии метафайла. Обычно это 0x00000001.

Checksum контрольная сумма данных метафайла.

Flags всегда содержит 0.

Size Размер данных метафайла в байтах.

BeginGroup / EndGroup комментарии содержат один или несколько EMF объектов. **BeginGroup** указывает начало списка, EMF записей, и заканчивается EndGroup структурой. Такие группы могут быть вложенными.

```
typedef struct _GdiCommentBeginGroup
{
   DWORD Identifier;       /* ID комментария(0x43494447) */
   DWORD Comment;          /* BeginGroup ID (0x00000002) */
   LONG  BoundsLeft;       /* Левая точка  прямоугольника */
   LONG  BoundsRight;      /* Правая точка прямоугольника */
   LONG  BoundsTop;        /* Верхняя точка прямоугольника */
   LONG  BoundsBottom;     /* Нижняя точка прямоугольника */
   DWORD SizeOfDescrip;    /* Количество символов в описании */
} GDICOMMENTBEGINGROUP;
```

Identifier содержит значение 0x43494447 идентифицирующее комментарий

Comment содержит значение 0x00000002 идентифицирующее структуру **BeginGroup** комментария.

BoundsLeft, BoundsRight, BoundsTop, и *BoundsBottom* определяют область вывода изображения.

SizeOfDescrip количество Unicode символов строки описания. Строка,если она присутствует, должна следовать сразу за заголовком. Поле равно 0, если строки нет.

EndGroup содержит только идентификационный заголовок без данных:

```
typedef struct _GdiCommentEndGroup
{
   DWORD Identifier;       /* комментария ID (0x43494447) */
   DWORD Comment;          /* EndGroup ID (0x00000003) */
```

```
} GDICOMMENTENDGROUP;
```

Identifier содержит значение 0x43494447 идентифицирующее комментарий.

Comment содержит значение 0x00000003 идентифицирующее структуру **EndGroup** комментария.

Multiformats комментарий используется для хранения метафайлов и Encapsulated PostScript (EPS) данных. Этот комментарий начинается с заголовка, затем идет одно или последовательност изображений:

```
typedef struct _GdiCommentMultiFormats
{
   DWORD Identifier;       /* ID комментария(0x43494447) */
   DWORD Comment;          /* Multiformats ID (0x40000004) */
LONG  BoundsLeft;       /* Левая точка  прямоугольника */
   LONG  BoundsRight;      /* Правая точка прямоугольника */
   LONG  BoundsTop;        /* Верхняя точка прямоугольника */
   LONG  BoundsBottom;     /* Нижняя точка прямоугольника */
   DWORD NumFormats;       /* Количество объектов в структуре */
   EMRFORMAT Data[];       /* Массим объектов */
} GDICOMMENTMULTIFORMATS
```

Identifier содержит значение 0x43494447 идентифицирующее комментарий.

Comment содержит значение 0x40000004 идентифицирующее структуру **Multiformats** комментария.

BoundsLeft, BoundsRight, BoundsTop, и *BoundsBottom* определяют область вывода изображения.

NumFormats кличество структур EMRFORMAT.

Data сами структуры EMRFORMAT.

```
typedef struct _EmrFormat
{
   DWORD Signature;    /* Идентификатор формата */
   DWORD Version;      /* Версия формата */
   DWORD Data;         /* Размер данных в байтах */
   DWORD OffsetToData; /* Смещение в байтах */
} EMRFORMAT;
```

Signature содержит значение 0x464D4520 идентифицирующее структуру метафайла, и значение 0x46535045 идентифицирующее Encapsulated PostScript файл.

Version версия данных. Для EPS содержит версию EPS файла. Для EMF обычно равно 0x00000001.

Data размер содержащейся структуры в байтах.

OffsetToData содержит смещение данных относительно начала структуры GDICOMMENTMULTIFORMATS.

Microsoft Windows Metafile Format (WMF) используется для хранения векторных и растровых изображений и графических данных в памяти или в дисковых файлах. Векторные данные хранимые WMF-файлом описывают команды Microsoft Windows Graphics Device Interface (GDI). Система интерпретирует и воспроизводит эти команды в контексте отображения, используя Windows API функцию PlayMetaFile(). Растровые изображения в WMF файле могут содержаться в формате Microsoft Device Dependent Bitmap (DDB), или Device Independent Bitmap (DIB).

В Windows метафайлы обычно создаются и воспроизводятся в памяти. Если данные метафайла слишком велики, что бы держать их в памяти, или должены быть сохранены прежде, чем приложение завершилось, то они могут быть записаны на диск в формате WMF или EMF и воспроизведены обратно с диска. Максимальный размер метафайла – 4 Гб.

WMF – это базовый 16-битный формат, который появлялся в Windows 2.0. Формат *EMF* является 32-битной дополненной переработкой формата

WMF. EMF расширил функциональное назначение WMF, включая цветную палитру и полную поддержку для всех 32-битовых команд GDI. Win32 API (Windows XP и Windows 7) и 32-битный OLE поддерживает как WMF, так и EMF файлы. Win16 API и 16-битный OLE поддерживает только WMF.

Хотя формат Windows Metafile поддерживают большое количество приложений, работающих на других платформах, основное его назначение – это обмен графической информацией между Windows приложениями. Например, Adobe's Encapsulated PostScript (EPS) поддерживает использование включенного Windows Metafile, когда требуется сохранить векторные данные.

При использовании метафайлов в Windows или OS/2, нет необходимости писать специальные программы для выполнения грамматического разбора их содержимого, можно вызывать имеющиеся в Windows API функции.

Все структуры данных и определения типов данных связанные с файлами WMF находятся в заголовочном файле WINDOWS.H. Для Win32 SDK определения WMF и EMF содержатся в WINUSER.H WINGDI.H. Оба этих SDK пригодны для всех C и C++ компиляторов, которые поддерживают разработку Windows приложений.

Список использованных источников

1. Роджерс, Д. Математические основы машинной графики / Д. Роджерс, Д. М. Адамс. - М.: Мир, 2001. - 606 с.

2. Роджерс, Д. Алгоритмические основы машинной графики / Д. Роджерс. - М.: Мир, 1989. - 512 с.

3. Павлидис, Т. Алгоритмы машинной графики и обработки изображений /Т. Павлидис. - М.: Радио и связь, 1986. - 398 с.

4. Аммерал, Л. Машинная графика на персональных компьютерах: пер. с англ. / Л. Аммерал. - М.: СолСистем, 1992. - 232 с

5. Аммерал, Л. Интерактивная трехмерная машинная графика: пер. с англ. / Л. Аммерал. - М.: СолСистем, - 317 с.

6. Аммерал, Л. Программирование графики на Турбо СИ: пер. с англ. / Л. Аммерал. - М.: СолСистем, - 203с.

7. Ньюмен, У. Основы интерактивной машинной графики / У. Ньюмен, Р. Спрулл. - М.: Мир, 1976. - 253 с.

8. Фокс, А. Вычислительная геометрия. Применение в проектировании и на производстве / А. Фокс, М. М. Прат. - М.: Мир, 1982. - 304 с.

9. Горельская, Л.В. Компьютерная графика: учеб. пособие / Л. В. Горельская , А. В. Кострюков , С. И. Павлов. - Оренбург: ИПК ОГУ, 2001. - 145с.

10. Ефимов, Н. В. Линейная алгебра и многомерная геометрия / Н. В. Ефимов, Э. Р. Розендорн. - М.: Наука, 2002. - 528 с.

11. Шикин, Е. В. Кривые и поверхности на экране компьютера / Е. В. Шикин, А. И. Плис. - М.: Диалог-МИФИ, 1996. - 240 с.

12. Иванов, В. П. Трехмерная компьютерная графика / В. П. Иванов, А.С. Батраков. - М.: Радио и связь, 1995. - 224 с.

13. Эгрон, Ж. Синтез изображений. Базовые алгоритмы / Ж. Эгрон. - М.: Радио и связь, 1993. -216 с.

14. 5. Lengyel, E. Mathematics for 3D Game Programming & Computer Graphics. 2nd.ed. Ed.David Pallai / E. Lengyel Charles River Media,Inc., 2004. - 348 p.