Федеральное агентство по образованию
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ГОУВПО «АмГУ»

УТВЕРЖДАЮ
Зав. кафедрой ТиЭФ
_____Е.А. Ванина
«_____»_____2007

КОМПЬЮТЕРНАЯ ГРАФИКА
УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ПО ДИСЦИПЛИНЕ

для специальности 010700 – «физика»

Составитель: к.ф.-м.н,
Стукова Е.В.

Благовещенск
2007

Печатается по решению
редакционно-издательского
совета инженерно-физического
факультета Амурского
государственного
университета

Е.В. Стукова

Учебно-методический комплекс по дисциплине «Компьютерная графика» для студентов очной формы обучения специальности 010700 – «Физика». – Благовещенск: Амурский гос. ун-т, 2007. – 94с.

Содержание

Федеральное агентство по образования Российской Федерации

Государственное учреждение высшего профессионального образования

АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

(ГОУ ВПО «АмГУ»)

“УТВЕРЖДАЮ”

Проректор по Учебно-научной работе

_____ Астапова Е.С..

“___”_____200__г.

РАБОЧАЯ УЧЕБНАЯ ПРОГРАММА

Дисциплина - Компьютерная графика.

для специальности 010701 - "Физика»

курс 5, семестр 9	
лекции	28 час.
практические занятия	нет
семинарские занятия	нет
лабораторные работы	28 час.
самостоятельная работа	28 час
консультация	3 час.
зачет	9 семестр
всего аудиторной нагрузки	64 час

Рабочая программа авторская

Составитель, к. ф.-м.н. доцент А.В. Бушманов
Факультет Инженерно-физический.
Кафедра теоретической и экспериментальной физики

2006 г.

Рабочая программа составлена на основании Государственного образовательного стандарта высшего профессионального образования и авторских разработок по направлению специальности) 010701

Рабочая программа обсуждена на заседании кафедры

“____”_____200 протокол №___

Зав. кафедрой_____( )

Рабочая программа одобрена на заседании УМСС 01.07.01

“___”_____200 протокол №___

Председатель УМС_____(_____)

СОГЛАСОВАНО
Начальник УМУ
_____Г.Н.Торопчина

СОГЛАСОВАНО
Председатель УМС факультета
_____

СОГЛАСОВАНО
Заведующий выпускающей кафедры
_____ ( )

“____”_____200_ г.

Рабочая программа переутверждена на 200__/200__ учебный год на заседании кафедры “___”_____ протокол № ___

Зав. выпускающей кафедры _____(_____)

Председатель УМСС _____(_____)

Проректор по УР _____(_____)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Программа курса "Компьютерная графика" входит в цикл общепрофессиональных дисциплин - Интегрированные системы автоматизированного управления, специальности 2202 - АСОИУ.

Цель курса - ознакомить с такими основными понятиями и методами компьютерной графики, как растровые алгоритмы, геометрические сплайны методы удаление скрытых линий и поверхностей, закрашивание, трассировка лучей, излучательность.

По завершению курса "Компьютерная графика", студент должен получить устойчивые навыки и умения:

1. построения графических примитивов в языках программирования;
2. работе с основными графическими устройствами;
3. преобразованиям на плоскости и в пространстве;
4. по разработке растровых алгоритмов;
5. по разработке геометрических сплайнов.

СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

ЛЕКЦИИ (28 часов)

1. Области применения компьютерной графики (2 часа):

тенденции построения современных графических систем; графическое ядро, инструментарий для написания приложений.

2. Машинная графика, обработка изображений и распознавание образов

виды данных представленных в форме изображений; классы изображений; координатная и растровая графика; буфер кадра; знакогенератор.

3. Примитивы вывода и атрибуты (2 часа):

зависимые атрибуты; векторный примитив; точечный примитив; текстовый примитив; растровый примитив; обобщенный примитив вывода; атрибуты примитивов вывода.

4. Системы координат и преобразования (2 часа):

мировые координаты; нормированные координаты; координаты устройства; преобразования сегментов.

5. Архитектура графических терминалов и графических рабочих станций абстрактная графическая рабочая станция; направления вывода на графические станции; типы графических станций; преобразование станции; доступ к нестандартным возможностям станции. Типы графических устройств запоминающие ЭЛТ с прямым сканирова-

нием изображения; векторные дисплеи с регенерацией изображения; растровые сканирующие дисплеи с регенерацией; центральное процессорное устройство; цифровой аналоговый преобразователь; системы с телевизионным растром.

6. Графические метафайлы (2 часа):

содержание метафайла; формат записей метафайла; стандарт метафайла ISO; элементы базового метафайла; уровни метафайлов.

7. Алгоритмы вычерчивания отрезков (2 часа):

пошаговый метод; цифровой дифференциальный анализатор; алгоритм Брезенхема; алгоритм Брезенхема генерации окружности.

Растровая развертка как способ генерации изображения (2 часа):

растровая развертка в реальном времени; список активных ребер (САР). групповое кодирование; клеточное кодирование.

8. Буферы кадра (2 часа):

сдвиговые регистры; архитектура буфера кадра; адресация растра; изображение литер; изображение отрезков.

9. Растровая развертка сплошных областей (2 часа):

растровая развертка; затравочное заполнение; растровая развертка многоугольников; простой алгоритм с упорядоченным списком ребер.

10. Алгоритмы заполнения (2 часа):

алгоритм заполнения по ребрам; алгоритм заполнения с затравкой; внутренне - заполняющие алгоритмы; гранично–заполняющие алгоритмы; простой алгоритм заполнения с затравкой.

11. Основы методов устранения ступенчатости (2 часа):

методы устранения искажений изображения; простой метод устранения лестничного эффекта; модифицированный алгоритм Брезенхема с устранением ступенчатости для первого квадранта.

12. Отсечение (2 часа):

двумерное отсечение; алгоритм отсечения Сазерленда – Коэна, основанный на разбиении отрезка; алгоритм разбиения средней точкой; трехмерное отсечение; трехмерный алгоритм разбиения средней точкой.

13. Определение выпуклости трехмерного тела и вычисление внутренних нормалей к его граням (2 часа):

разрезание невыпуклых тел; отсечение многоугольников; отсечение литер.

14. Удаление невидимых линий и поверхностей (2 часа):

алгоритм плавающего горизонта; алгоритм Варнока; алгоритм разбиения криволинейных поверхностей. Алгоритм, использующий z – буфер алгоритмы использующие список приоритетов; интервальный алгоритм построчного сканирования; определение видимых поверхностей путем трассировки лучей.

Лекция 1. Области применения компьютерной графики

Направления компьютерной графики

В нынешнем, устоявшемся состоянии принято разделять компьютерную графику на следующие направления:

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

Изобразительная компьютерная графика

Объекты: синтезированные изображения.

Задачи:

- построение модели объекта и генерация изображения,
- преобразование модели и изображения,
- идентификация объекта и получение требуемой информации.

Обработка и анализ изображений

Объекты: дискретное, числовое представление фотографий.

Задачи:

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

Итак, в основе обработки и анализа изображений лежат методы представления, обработки и анализа изображений плюс, естественно, изобразительная компьютерная графика хотя бы для того, чтобы представить результаты.

Анализ сцен

Предмет: исследование абстрактных моделей графических объектов и взаимосвязей между ними. Объекты могут быть как синтезированными, так и выделенными на фотоснимках.

Первый шаг в анализе сцены - выделение характерных особенностей, формирующих графический объект(ы).

Примеры: машинное зрение (роботы), анализ рентгеновских снимков с выделением и отслеживанием интересующего объекта, например, сердца.

Итак, в основе анализа сцен (перцептивной компьютерной графики) находятся изобразительная графика + анализ изображений + специализированные средства.

Когнитивная компьютерная графика

Только формирующееся новое направление, пока недостаточно четко очерченное.

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

Общая последовательность познания заключается в, возможно циклическом, продвижении от гипотезы к модели (объекта, явления) и решению, результатом которого является знание (рис. 1).

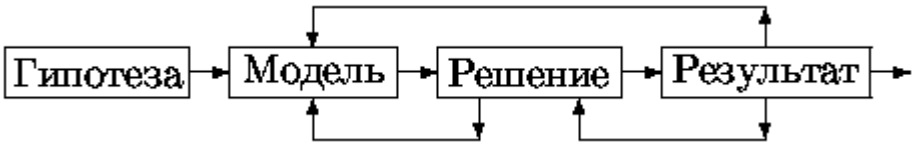


Рис. 1.Последовательность процесса познания

Человеческое познание использует два основных механизма мышления, за каждым из которых закреплена половина мозга:

- осознанное, логико-вербальное, манипулирует абстрактными последовательностями символов (объектов) + семантика символов + праг-

матические представления, связанные с символами. Возраст этого механизма, связанного с наличием речи, - до 100 тыс. лет:

- неосознанное, интуитивное, образное, работает с чувственными образами и представлениями о них. Возраст этого механизма - время существования на Земле животного мира.

Первоначально ЭВМ имели малую производительность процессоров и средств компьютерной графики, т.е. по сути дела имели возможность работы только с символами (некоторый упрощенный аналог логического мышления).

С появлением супер-ЭВМ, производительностью в миллиард и более операций в секунду и графических супер-станций, производительностью до сотен миллионов операций в секунду, появилась возможность достаточно эффективного манипулирования образами (картинами).

Важно отметить, что мозг не только умеет работать с двумя способами представления информации, причем с образами он работает иначе и эффективнее чем ЭВМ, но и умеет соотносить эти два способа и совершать (каким-то образом) переходы от одного представления к другому.

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

Другие важнейшие задачи:

- визуализация тех знаний, для которых не существует (пока ?) символических описаний,
- поиск путей перехода от образа к формулировке гипотезы о механизмах и процессах, представленных этими (динамическими) образами на экране дисплея.

Появление когнитивной компьютерной графики - сигнал о переходе от эры экстенсивного развития естественного интеллекта к эре интенсивного развития, характеризующегося глубоко проникающей компьютеризацией, рождающей человеко-машинную технологию познания, важным моментом которой является непосредственное, целенаправленное, активирующее воздействие на подсознательные интуитивные механизмы образного мышления.

Одним из ярких и наиболее ранних примеров приложения когнитивной компьютерной графики является работа Ч. Страуса "Неожиданное применение ЭВМ в чистой математике" (ТИИЭР, т. 62, N 4, 1974, с. 96 - 99). В ней показано как для анализа сложных алгебраических кривых используется "n-мерная" доска на основе графического терминала. Пользуясь устройствами ввода математик может легко получать геометрические изображения результатов направленного изменения параметров исследуемой зависимости. Он может также легко управлять текущими значениями параметров, "углубляя тем самым свое понимание роли вариаций этих параметров". В результате получено "несколько новых теорем и определены направления дальнейших исследований".

Приложения компьютерной графики

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

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

- средства массовой информации,
- досуг.

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

По телевидению часто можно видеть передачи иллюстрирующие приложения компьютерной графики в автоматизации проектирования (были передачи об автоматизированном проектировании самолетов, автомобилей), много передач об автоматизации производства с различными робототехническими системами.

Передачи о мире бизнеса практически не обходятся без показа различной дисплейной техники и ее использования.

Что касается искусства, то достаточно упомянуть, что один из самых крупных первых суперкомпьютерных центров мира находился на студии Уолта Диснея и использовался для подготовки мультфильмов. Всем известно, что многие "жутики" и боевики также готовились с широким использованием средств компьютерной графики для подготовки высокореалистичных сцен.

Применение компьютерной графики в средствах массовой информации мы видим ежедневно, как в виде различных заставок и телеэффектов на экране, так и в виде газет, при подготовке многих из которых используется электронная верстка на компьютере.

С компьютерными играми, отнимающими не только время досуга, конечно же знаком каждый.

Поэтому здесь мы рассмотрим, в основном, приложения компьютерной графики в компьютерном моделировании, а также немного познакомимся с самым новым приложением - системами виртуальной реальности.

Компьютерная графика в интегрированной САПР

Общий цикл разработки какого-либо промышленного изделия, будь то радиатор батареи центрального отопления или же самолет можно представить в виде схемы, показанной на рис. 2.

Рис. 2. Общий цикл разработки изделия

Естественно, что могут выполняться возвраты на повторное выполнение общего цикла, начиная с некоторого этапа.

Компьютерная графика в компьютерном моделировании

Ясно, что в настоящее время на всех этапах разработки в той или иной мере используются ЭВМ и, следовательно, компьютерная графика. На этапе исследований важную роль, кроме натурных экспериментов, играет компьютерное моделирование работы изделия и/или его компонент. Рассмотрим в качестве примера задачи моделирования быстропротекающих процессов механики сплошной среды. К таким задачам относятся задачи сварки и упрочнения взрывом, задачи расчета синтеза алмазов взрывом, задачи расчета защиты космических аппаратов от метеоритов и микрометеоритов и т.д.

Следует отметить что реальный физический процесс развивается за единицы или десятки микросекунд, поэтому натурный эксперимент не только дорогостоящ, но и дает мало информации. Обычно это один или несколько рентгеновских снимков, прямые измерения, как правило, или затруднены, или датчики вносят заметные изменения в процесс.

В этой связи опыт специалистами в этой области набирается довольно долго - порядка десятка и более лет и зачастую носит интуитивный характер. Поэтому одна из важнейших функций компьютерного моделирования - быстрое получение опыта. Ведь по сути дела комплекс компьютерного моделирования при адекватности моделей, заложенных в него, представляет собой уникальную экспериментальную установку, на которой можно измерить (выдать на дисплей) любую величину, что угодно изменить, даже приостановить

процесс для подробного анализа и пустить дальше. Попробуйте это сделать со взрывом!

Другая важная функция комплекса компьютерного моделирования, особенно при оснащении его средствами коллективного, управляемого просмотра машинных фильмов, показывающих процесс не отдельными картинками, а в динамике - унификация представлений о процессе у заинтересованных специалистов - исследователей, конструкторов, технологов, испытателей. Так как обычно, при использовании только числовой информации и без образного ее представления у разных специалистов формируются собственные представления (не обязательно у всех одинаковые и правильные).

Лекция 2. Машинная графика, обработка изображений и распознавание образов

Для представления графической информации на двумерной плоскости (например, экране монитора, странице книги и т.п.) в вычислительной технике применяются два основных подхода: растровый и векторный[1)].

При векторном подходе графическая информация описывается как совокупность неких абстрактных геометрических объектов, таких как прямые, отрезки, кривые, прямоугольники и т.п.

Растровая графика же оперирует изображениями в виде **растров**. Неформально можно сказать, что растр - это описание изображения на плоскости путем разбиения всей плоскости или ее части на одинаковые квадраты и присвоение каждому квадрату своего цветового (или иного, например, прозрачности, для последующего наложения изображений друг на друга) атрибута. Если таких квадратов имеется конечное число, то получается, что непрерывная цветовая функция изображения приближенно представлена конечной совокупностью значений атрибутов. Иногда понятие растра определяют более широко: как разбиение плоскости (или ее участка) на равные элементы (т.е."замощение"), например шестиугольниками (**гексагональный** растр). Далее в этой книге расширенное толкование использоваться не будет.

С другой стороны, растр можно рассматривать как кусочно-постоянную аппроксимацию изображения, заданного как цветовая функция на плоскости. Такая точка зрения позволяет применять математический аппарат теории аппроксимации для работы с растровыми изображениями, о чем подробнее будет рассказано далее.

Формально, введем следующие определения:

Растр (англ. raster) - отображение вида $f: X \times Y \to 2^{R^2} \times C$, где $X \subset Z$, $Y \subset Z$, 2^{R^2}обозначает множество всех подмножеств R^2, C - множество значений атрибутов (как правило, цвет). f(i, j) - элемент растра, называемый **пикселем** (англ. pixel (от picture element)), в русскоязычной литературе иногда также переводится как пиксел);

f(i, j) = (A(i, j),C(i, j)), где

- $A(i,j) \subset R^2$- область пикселя,
- $C(i,j) \in C$- атрибут пикселя (как правило, цвет). Чаще всего мы будем пользоваться следующими двумя видами атрибутов:
  - C(i, j) = I(i, j) - интенсивность (или яркость) пикселя;
  - C(i, j) = {R(i, j),G(i, j),B(i, j)} - цветовые атрибуты в цветовой модели RGB.

Также иногда будут употребляться матричные обозначения:

Mij = (Aij ,Cij)

Aij может определяться двояко, в зависимости от того, с какой моделью мы хотим работать:

- Aij := (i, j) - одна точка. Пример такой модели растра см. на рис. 1;
- $Aij := (i, i+1) \times (j, j+1)$- квадрат. Пример такой модели растра см. на рис. 1.2. На реальных графических устройствах физически пиксели могут быть прямоугольниками, что иногда порождает дополнительные трудности.

В реальности, как правило, X и Y - ограниченные наборы неотрицательных целых чисел; такой растр называется **прямоугольным**. Для него

применимо понятие **Аспектовое отношение** (англ. aspect ratio) - отношение ширины к высоте растра (|X|/|Y|). Чаще всего такое понятие употребляется в связи с физическими растрами (дисплеями, ПЗС-матрицами фотоаппаратов и т.д.) и записывается в виде простой дроби с ":", например "4:3".

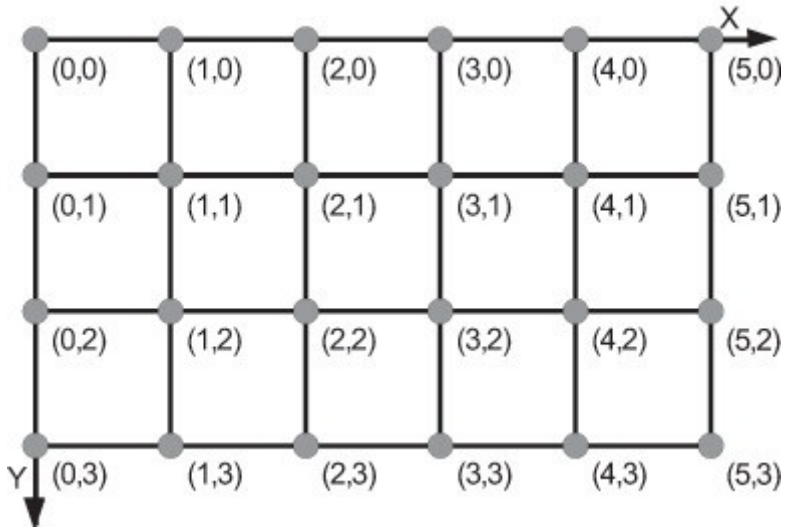


Рис. 1. Модель растра первого типа.

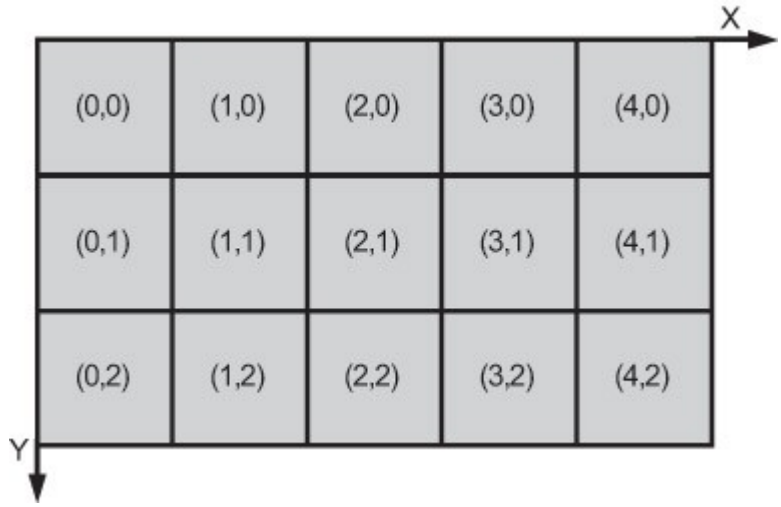


Рис. 2. Модель растра второго типа.

Бесконечные растры (когда X и Y неограниченны) бывают удобны для описания алгоритмов, позволяя избежать особых ситуаций. Впрочем, самой сутью некоторых алгоритмов является как раз работа с граничными случаями.

Растровое представление является естественным в тех случаях, когда нам не известна дополнительная информация об изображаемых объектах (например, цифровым фотоаппаратом можно снимать изображения произ-

вольного содержания). В случае же векторного описания примитивами являются более сложные объекты (линии и области, ограниченные линиями), что предполагает априорные знания о структуре изображения. В последнее время проявляется ярко выраженная тенденция к преобладанию устройств ввода-вывода двумерной графической информации, основанных на растровом принципе как более универсальном. Возникающая при выводе задача отображения геометрических объектов, заданных их математическим описанием (например, координатами концевых точек и цветом для отрезка), на растре, называемая **растеризацией**, рассмотрена в последующих разделах.

При построении алгоритмов, работающих с изображениями, можно также пользоваться информацией как непосредственно атрибутов пикселей, так и работать с примитивами более высокого порядка. В данной книге в основном рассматриваются алгоритмы первого типа, про которые говорят, что они работают в **пространстве изображения** (англ. image space), тогда как вторые работают в **объектном пространстве** (англ. object space) (эти термины чаще употребляются в трехмерной графике).

Синтез изображения

Синтез изображения при помощи компьютера можно представить в виде последовательности шагов, хотя реально эти шаги в программе часто переплетаются:

- Задание объекта;
- Удаление скрытых поверхностей;
- Закраска.

Объекты можут быть заданы, как:

1. Двумерные объекты.

Они моделируются при помощи таких примитивов, как:

отрезки (заданные двумя конечными точками), многоугольники (определенные списком вершин и, возможно, заполняющим узором), окружности (опи-

сываемые центром, радиусом и, возможно, заполняющим узором), а также полиномиальные кривые (заданные своими коэффициентами).

2. Трехмерные объекты.

В случае трехмерных измерений соответствующие примитивы определяются путем добавления координаты z.

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

Системы моделирования тел порождают трехмерные объекты, основываясь на:

1. интерактивном задании параметров (при взаимодействии с пользователем);
2. либо автономном.

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:

- объединение (соединение двух объектов);
- пересечение (выделение общего подмножества);
- 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.

Удаление скрытых поверхностей

Скрытая поверхность - поверхность или ее часть, невидная из точки, где находится воображаемая камера; поверхности, представляющие дальние стороны объектов или поверхности, закрытые другими, более близкими поверхностями.

Различные методы удаления невидимых поверхностей могут быть реализованы аппаратно. В алгоритмах обычно предполагается, что экран расположен в проекционной плоскости z = 0, а сцена находится позади него.

Существуют следующие алгоритмы удаления поверхностей:

- алгоритм z-буфера;

В этом алгоритме имеется отдельный буфер значений координаты z, по одному значению на каждый пиксел. Величина z пиксела содержит информацию о глубине соответствующей точки, которая проецируется на пиксел и принадлежит ближайшему из встречающихся к данному моменту многоугольников. Когда новый многоугольник преобразуется, усекается и проецируется на плоскость z = 0, значения пикселов, составляющих многоугольник, по очереди сравниваются со значениями, записанными в z-буфере. Если зна-

чение z пиксела данного многоугольника оказывается меньше (это означает, что данный многоугольник расположен ближе к экрану, чем любой из ранее рассмотренных многоугольников), то пиксел считается текущим "видимым" пикселом и запоминается как в буфере регенерации, так и в z-буфере. Реально же он станет видимым только в том случае, если не заменится к тому времени, когда будет обработан последний многоугольник.

- алгоритм "художника";

В отличии от алгоритма z-буфера, который может обрабатывать многоугольники, поступающие в произвольном порядке, в алгоритме "художника" многоугольники сначала упорядочиваются в направлении от заднего плана к переднему. В случае когда пары многоугольников не удается достаточно просто упорядочить, они подразделяются на части до тех пор, пока получившиеся части не позволят это сделать. Затем многоугольники проецируются и "раскрашиваются" в буфере кадров в порядке от заднего плана к переднему так, чтобы многоугольники, находящиеся ближе к точке наблюдения, правильно закрывали более удаленные многоугольники и при этом не требовались дополнительные вычисления.

Закраска

В модели закраски должны приниматься во внимание как **свойства поверхности** (ее цвет, фактура и отражательная способность), так и **относительное местоположение, ориентация и свойства источников света и других поверхностей**. В моделях освещения для световых источников могут учитываться такие варианты, как рассеянный свет, точечные источники (солнце или ярко горящая лампа) или распределенные источники (окно или ряд флуоресцентных трубок).

Рассеянный свет моделируется путем добавления постоянной величины световой интенсивности ко всем поверхностям, однако, при такой стратегии не проводится никакой дифференциации между поверхностями. Отражение точечных источников света матовыми или диффузными поверхностями

(т.е. такими, которые рассеивают свет равномерно по всем направлениям) описывается законом косинусов Ламберта.

Алгоритмы построения теней от источников света напоминают алгоритмы удаления скрытых поверхностей, поскольку в них определяется, какие поверхности могут быть "видимы" из точек, где расположены источники света. Поверхности, одновременно видимые из точки наблюдения и из источников света, не находятся в тени, в то время как те из них, которые видимы из точки наблюдения, но не видимы из светового источника, лежат в тени. В случае распределенных источников света нужны сложные вычисления, как для полных теней, так и для полутеней.

Еще более сложную проблему создает фактор пропускания света. "Направленное" прохождение, характерное для прозрачных поверхностей, подобных стеклу, определяется коэффициентом преломления вещества. Диффузное прохождение света через полупрозрачные материалы, такие как замерзшее стекло, приводит к рассеянию света по всем направлениям.

Алгоритм трассировки лучей позволяет строить наиболее реалистичные изображения с учетом, как отражения, так и преломления света. В таких алгоритмах отслеживаются индивидуальные световые лучи, чтобы определить, какие из них оканчиваются в точке наблюдения и как они оказываются там. Для того, чтобы не иметь дела с бесконечным числом лучей, исходящих из источника света, лучи прослеживают в обратном направлении, с началом в каждом пикселе. Всякий луч, выходящий из точки наблюдения и проходящий через пиксел, проецируется назад до пересечения с поверхностью. Затем обратное трассирование отраженного луча продолжается, чтобы определить, пришел ли этот луч от источника света или он является результатом отражения от другого объекта. В случае прозрачной поверхности должен быть отслежен и второй, преломленный луч. Каждый луч необходимо проверить на пересечение с каждым объектом.

Для формирования **фактуры поверхности** могут использоваться различные модели, вносящие локальные нерегулярности. Для отображения двумерного узора на поверхность можно воспользоваться узором, образованным значениями интенсивности, чтобы промодулировать интенсивности, вычисленные при помощи алгоритмов закраски и затенения.

Общая характеристика задач распознавания образов и их типы.

Под образом понимается структурированное описание изучаемого объекта или явления, представленное вектором признаков, каждый элемент которого представляет числовое значение одного из признаков, характеризующих соответствующий объект. Общая структура системы распознавания и этапы в процессе ее разработки показаны на рис. 3.

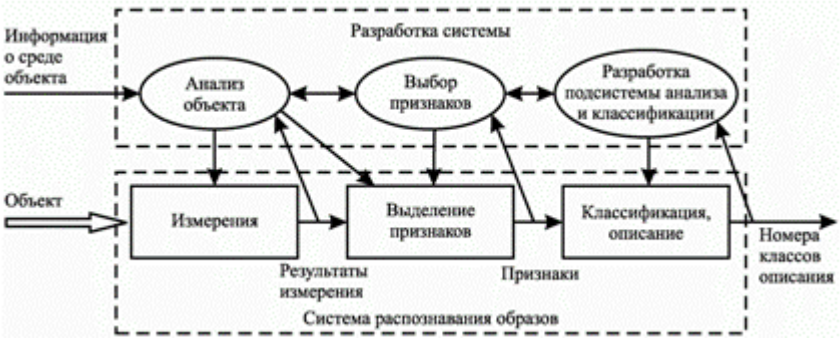


Рис. 3. Структура системы распознавания

Суть задачи распознавания - установить, обладают ли изучаемые объекты фиксированным конечным набором признаков, позволяющим отнести их к определенному классу.

Задачи распознавания имеют следующие характерные черты.

1. Это информационные задачи, состоящие из двух этапов: а) приведение исходных данных к виду, удобному для распознавания; б) собственно распознавание (указание принадлежности объекта определенному классу).
2. В этих задачах можно вводить понятие аналогии или подобия объектов и формулировать понятие близости объектов в качестве основания для зачисления объектов в один и тот же класс или разные классы.

3. В этих задачах можно оперировать набором прецедентов-примеров, классификация которых известна и которые в виде формализованных описаний могут быть предъявлены алгоритму распознавания для настройки на задачу в процессе обучения.
4. Для этих задач трудно строить формальные теории и применять классические математические методы (часто недоступна информация для точной математической модели или выигрыш от использования модели и математических методов не соизмерим с затратами).
5. В этих задачах возможна "плохая" информация (информация с пропусками, разнородная, косвенная, нечеткая, неоднозначная, вероятностная).

Целесообразно выделить следующие типы задач распознавания.

1. Задача распознавания - отнесение предъявленного объекта по его описанию к одному из заданных классов (обучение с учителем).
2. Задача автоматической классификации - разбиение множества объектов (ситуаций) по их описаниям на систему непересекающихся классов (таксономия, кластерный анализ, обучение без учителя).
3. Задача выбора информативного набора признаков при распознавании.
4. Задача приведения исходных данных к виду, удобному для распознавания.
5. Динамическое распознавание и динамическая классификация - задачи 1 и 2 для динамических объектов.
6. Задача прогнозирования - это задачи 5, в которых решение должно относиться к некоторому моменту в будущем.

Знакогенератор

Знаки (буквы, цифры, значки), наряду с векторами, являются наиболее широко употребимыми графическими примитивами. Они в общем представлены в закодированной форме, например, ASCII, КОИ-8 и т.д. Эти коды

преобразуются знакогенератором в координатные значения для представления соответствующих знаков.

Используются два основных метода - векторный метод и метод матрицы точек.

Лекция 4. Системы координат и преобразования

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. В данном разделе будут рассмотрены математические методы для описания геометрических преобразований координат в двух, трех и четырехмерном случае, будут обсуждены некоторые вопросы эффективности, рассмотрены геометрические преобразования растровых картин.

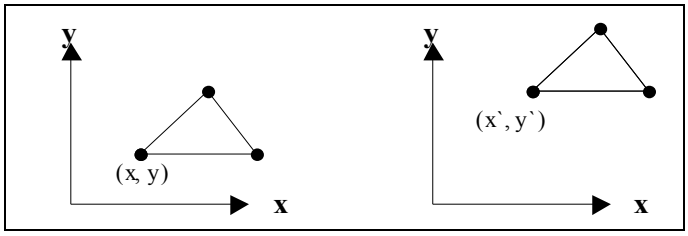
Далее большими буквами X, Y, Z будут обозначаться обычные декартовые координаты, а маленькие буквы x, y, z будут использоваться для обозначения т.н. однородных координат.

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. В данном разделе будут рассмотрены математические методы для описания геометрических преобразований координат в двух и трехмерном случае, будут обсуждены некоторые вопросы эффективности, рассмотрены геометрические преобразования растровых картин.

Далее большими буквами x, y, z будут обозначаться обычные декартовые координаты, а маленькие буквы X, Y, Z будут использоваться для обозначения т.н. однородных координат.

Двумерные геометрические преобразования

Параллельный перенос



Параллельный перенос в плоском случае имеет вид:

$$\begin{cases} x` = x + Dx \\ y` = y + Dy \end{cases}$$

$$\underbrace{[x', y']}_{P'} = \underbrace{[x, y]}_{P} + \underbrace{[Dx, Dy]}_{T}$$

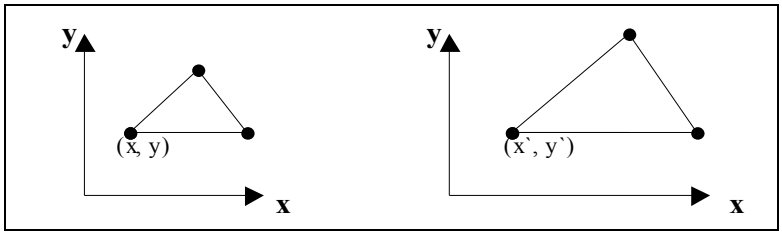
или в векторной форме:

$$\mathbf{P`} = \mathbf{P} + \mathbf{T},$$

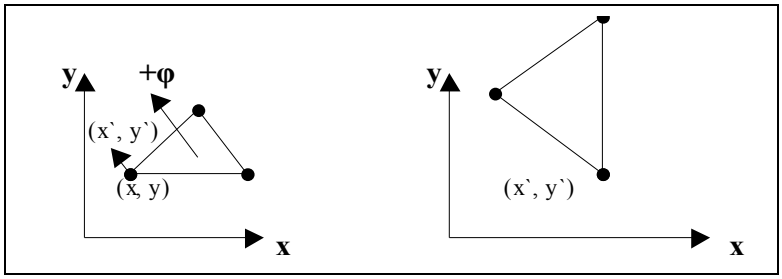
где $\mathbf{P`} = [x` \; y`]$ - вектор-строка преобразованных координат,

где x, y - исходные координаты точки, Tx, Ty - величина сдвига по осям, x`, y` - преобразованные координаты. $\mathbf{P} = [x \; y]$ -- вектор-строка исходных координат, $\mathbf{P`} = [x` \; y`]$ -- вектор-строка преобразованных координат, $\mathbf{T} = [Tx \; Ty]$ -- вектор-строка сдвига.

Масштабирование



Поворот



Преобразование в однородную систему координат

Как видно двумерные преобразования имеют различный вид. Сдвиг реализуется сложением, а масштабирование и поворот - умножением. Это различие затрудняет формирование суммарного преобразования и устраняется использованием двумерных однородных координат точки, имеющих вид:

[X

Y W].

Здесь W - произвольный множитель не равный 0.

Двумерные декартовые координаты точки получаются из однородных делением на множитель W:

$x = X / W, y = Y / W, W \neq 0$

Однородные координаты можно представить как промасштабированные с коэффициентом W значения двумерных координат, расположенные в плоскости с Z = W.

В силу произвольности значения W в однородных координатах не существует единственного представления точки, заданной в декартовых координатах.

Преобразования параллельного переноса, масштабирования и поворота в однородных координатах относительно центра координат все имеют одинаковую форму произведения вектора исходных координат на матрицу преобразования.

Лекция 5. Архитектура графических терминалов и графических рабочих станций

Здесь мы рассмотрим самые общие соображения по построению графической системы, не зависящие от метода формирования изображения (произвольное или растровое сканирование луча).

В общем, цель графической системы - визуализация модели объекта, обрабатываемой компьютером.

Можно представить себе два основных способа представления модели объекта - алгоритмический, когда модель объекта вместе с описанием изображения и процесса его генерации "размазаны" по некоторому алгоритму. Другой способ - представление модели объекта в структуре данных. В этом случае достигается разделение по сути дела различных функций - манипулирование моделью (моделирование объекта, сцены и т.п.) со стороны прикладной системы и отображение модели в виде картин.

Ясно, что отображение, хотя временами и очень трудная, но стабильная проблема, в то время как моделирование определяется потенциально бесконечным количеством приложений. Поэтому, оговорив интерфейс, есть смысл разделять моделирование и отображение так чтобы было возможно из независимое развитие.

Принципиальным моментом в разработке распределенной системы является выбор оптимального разделения функций между подсистемами моделирования и отображения. Естественно предполагать, что какая-либо граница раздела должна находиться в месте полного завершения выполнения некоторой функции. Кроме того, вряд ли целесообразно разделять между различными подсистемами некоторый файл данных и программу (процессор) его "интенсивной" обработки. Для уяснения задачи распределения функций рассмотрим модель графической системы с точки зрения процессов преобразования информации при формировании изображения.

Процесс вывода может быть представлен следующей схемой:

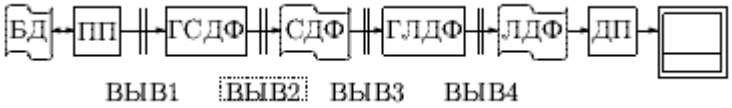


Рис.: Модель процесса вывода в графической системе. БД - база данных прикладной программы; ПП - прикладная программа; ГСДФ - генератор структурированного дисплейного файла; СДФ - структурированный дисплейный файл; ГЛДФ - генератор линейного дисплейного файла; ЛДФ - линейный дисплейный файл; ДП - дисплейный процессор; ВЫВ1, ВЫВ2, ВЫВ3, ВЫВ4 - возможные границы разделения функций

В этой модели отдана дань традиционному соображению о полезности структурированного дисплейного файла, представляющего собой иерархию вложенных объектов. Каждый объект, кроме примитивных элементов, может содержать экземпляры объектов нижнего уровня. Экземпляр объекта может сопровождаться набором совершаемых над ним преобразований. Использование экземпляров объекта выполняется либо его копированием, либо ссылкой (аналогично вызову подпрограммы). На самом же деле такое описание изображения подразумевает иерархичность его структуры, которая действительно имеет место в только ряде задач проектирования.

Итак: ВЫВ1 - подсистема моделирования готовит высокоуровневую информацию для построения структурированного дисплейного файла. Подсистема отображения формирует его и поддерживает работу с таким файлом, т.е. обмен информацией производится на наиболее высоком семантическом уровне.

ВЫВ2 - подсистема моделирования формирует структурированный дисплейный файл и передает его на хранение и интерпретацию в подсистему отображения. Понятно, что в этом случае возрастает разнообразие и сложность передаваемых данных, так как кроме информации по построению изображения должна передаваться информация, связанная с управлением структурой данных.

ВЫВ3 - подсистема моделирования формирует информацию для построения линейного дисплейного файла, возможно сегментированного. Линейный дисплейный файл формируется и поддерживается подсистемой отображения. Он является результатом выполнения геометрических преобразований, преобразований окно - порт и отсечения над полным изображением.

ВЫВ4 - подсистема моделирования формирует линейный (сегментированный) дисплейный файл и в аппаратно-ориентированной форме передает его подсистеме отображения, в задачи которой входит лишь поддержка функций приема и интерпретации файла в изображение дисплейным контроллером. Требования к вычислительной мощности подсистемы отображения и ее программному обеспечению в данном случае минимальны.

Следует отметить, что, так называемая, аппаратно-ориентированная форма для современных дисплеев, как правило, весьма высокоуровневая и включает в себя типичные наборы примитивов, атрибутов, средств сегментирования и даже структурирования. Аппаратность же в основном заключается в использовании некоторой вполне определенной формы кодирования, причем обычно наиболее компактной.

Из рассмотренного ясно, что наиболее целесообразны границы раздела ВЫВ1, ВЫВ3, ВЫВ4. Причем первые две обеспечивают аппаратную независимость, а последняя соответствует минимальным требованиям к подсистеме отображения.

Естественно полагать, что подсистема моделирования находится в ЦП, а подсистема отображения представлена дисплейным процессором.

Для повышения быстродействия при выполнении преобразований используются архитектуры с двойной буферизацией, использующие два дисплейных контроллера и два буфера для сохранения изображения. Дисплейный контроллер 1 выполняет преобразования и формирует линейный дисплейный файл в одном из буферов. Параллельно с этим более простым дис-

плейный контроллером 2 ведется отображение из второго буфера. Затем буфера переключаются.

Дисплеи с произвольным сканированием луча (каллиграфические или векторные дисплеи)

Первые графические векторные дисплеи с регенерацией появились на зарубежном рынке в конце 60-х годов. В нашей стране серийные векторные дисплеи появились в середине 70-х годов.

Практически с самого начала векторные дисплеи имели разрешение 1024×1024 точки и могли без мерцания отображать от 1500 до 3000 векторов. Векторы могли быть нескольких типов (сплошные, штриховые, точечные, пунктирные и т.д.) с несколькими градациями яркости. Встроенные средства генерации символов обычно имели возможность строить символы двух фиксированных размеров нескольких ориентаций. В качестве средства диалога использовались буквенно-цифровая и символьная клавиатуры и световое перо.

Рассмотрим архитектуру простой графической системы. Далее будет использоваться понятие "дисплейный процессор" широко распространенное в литературе, но не имеющее точного определения. Здесь под ним мы будем понимать аппаратную часть графической системы, которая:
 формирует изображение на дисплее из описания картины;
 обрабатывает графический ввод от пользователя.

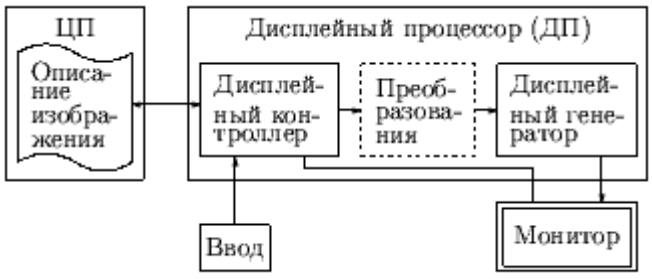


Рис. Общая архитектура простой графической системы

Обе задачи весьма специальны и вычислительно интенсивны и могли бы значительно затруднить обычную работу ЦП.

В зависимости от требований приложений и используемых приборов вывода дисплейные процессоры имеют различную сложность и работают на различных принципах. По принципу формирования изображения различаются два основных класса дисплейных процессоров:

 каллиграфические или векторные устройства, называемые еще устройствами с произвольным сканированием луча;

 растровые устройства.

Дисплейный процессор обязательно состоит из дисплейного контроллера и дисплейного генератора. Более мощные дисплейные процессоры оснащаются и собственными средствами преобразования и сохранения геометрических данных.

Рассмотрим назначение модулей дисплейного процессора справа-налево: дисплейный генератор, преобразования, дисплейный контроллер.

В данном разделе рассматривается архитектура дисплеев с произвольным и растровым сканированием луча, использующих ЭЛТ в качетве устройства отображения.

Вводятся понятия дисплейного файла, линейного дисплейного файла и структурированного дисплейного файла, дисплейного процессора.

Показывается, что дисплейный процессор состоит из дисплейного контроллера, блока преобразований и дисплейного генератора.

Рассматриваются два алгоритма работы генератора векторов - цифрового дифференциального анализатора и Брезенхема.

Анализируется процесс преобразования модели объекта в изображение, рассматриваются целесообразные границы разделения функций между центральным процессором и графической системой, описываются альтернативные архитектуры дисплеев.

В настоящее время подавляющее распространение получили растровые дисплеи. Рассмотрение векторных дисплеев производится, в основном, из-за того что построение изображения на них осуществляется с использованием привычных понятий позиционирования и вычерчивания. На примере таких устройств формулируются общие понятия и выделяются принципиальные компоненты любой графической системы, назначением которой является представление модели объекта в изображение.

В векторных дисплеях изображение строится в виде совокупности отдельно и достаточно точно выдаваемых отрезков. Основная проблема большинства дисплеев, в частности дисплеев на обычных электронно-лучевых трубках состоит в том, что если линия прочерчивается один раз, то за время послесвечения она пропадет с экрана. Решение этой проблемы заключается в том, что построение изображения циклически повторяется (регенерируется) с требуемой частотой (обычно с частотой сети - 50 Гц). Для такой регенерации используется дисплейный файл, представляющий собой описание изображения.

Лекция 7. Алгоритмы вычерчивания отрезков

Пусть у нас есть некоторая кривая, и мы хотим построить ее изображение на растровой решетке. Возникает вопрос: какие из ближайших пикселей следует закрашивать? В данной и следующей лекциях мы рассмотрим случай построения на монохромном растре, когда возможны только два уровня интенсивности закраски пикселя - "полностью закрашен" или "полностью не закрашен". Если же допустимы несколько уровней интенсивности, то можно растеризовывать более аккуратно, уменьшая эффекты алиасинга (т.е. ступенчатости).

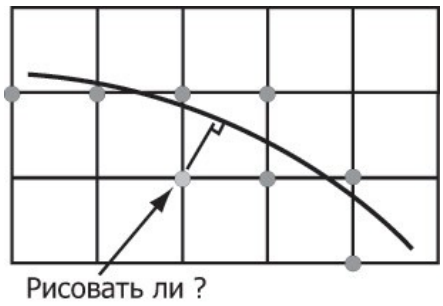
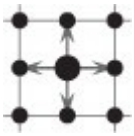


Рис. 1. Изображение кривых на растре.

Пусть (x_0, y_0) - фиксированный пиксель, а (x, y) - некоторый другой пиксель на плоскости. Тогда для определения их близости вводятся следующие понятия:

1. 4-связность

$|x-x_0|+|y-y_0|=1$



2. 8-связность

$\max\{|x-x_0|+|y-y_0|\}=1$

В дальнейших рассуждениях расстояние будем считать заданным стандартной евклидовой метрикой.

Изображение отрезка с целочисленными координатами концов

Пусть наш отрезок - это AB. Перейдем от системы координат Oxy к Ax'y'.Отрезок может лежать в любом из 8 октантов, но всегда существуют симметрии относительно осей, разделяющих эти октанты, симметрии определяются матрицами

$$\begin{pmatrix} \pm 1 & 0 \\ 0 & \pm 1 \end{pmatrix}$$

и

$$\begin{pmatrix} 0 & \pm 1 \\ \pm 1 & 0 \end{pmatrix},$$

позволяющие свести задачу к случаю отрезка, лежащего в первом октанте, в нем матрица имеет вид

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

Назовем такой случай каноническим, в дальнейшем будут рассмотрены алгоритмы для этого случая. В каноническом случае процесс рисования 8-связной линии можно закодировать последовательностью вида: sdssd... (рис. 3.3), где

- s - горизонтальное смещение;
- d - диагональное смещение.

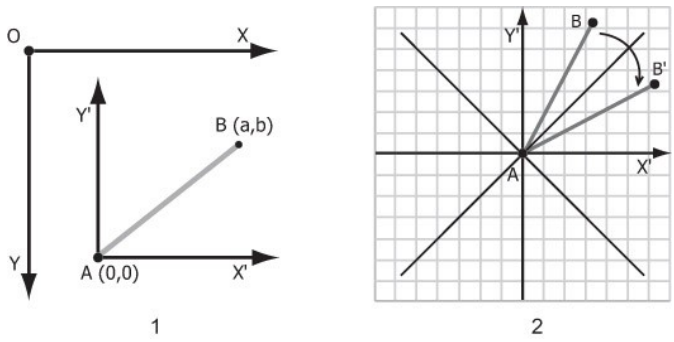


Рис. 2. Переход к каноническому случаю в два этапа.

Эквивалентно этой последовательности можно сопоставить бинарный код, где 0 соответствует s, а 1 соответствует d. Такой код для рисования отрезка $(0,0) \to (p,q), p > q \in N$ называется кодом Ротштейна для $\frac{p}{q}$.

Пусть plot(x,y) - функция, закрашивающая точку растра с координатами (x,y).

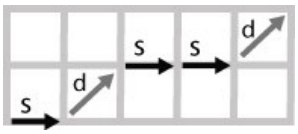


Рис. 3. Кодирование закрашивания отрезка (или код Ротштейна).

Цифровой дифференциальный анализатор

Алгоритм **Цифровой дифференциальный анализатор** (англ. DDA - Digital Differential Analyzer) строит 8-связную линию.

Для начала, пусть $P_1 = (0, 1)$; $P_2 = (1, 1)$. Для определения того, какой из пикселей, - P_1 или P_2, - следует закрасить, сравним расстояния до них. В силу подобия треугольников, образованных пересечением рисуемого отрезка, прямой x = 1 и перпендикулярами из P_1 и P_2 на отрезок (рис. 4), достаточно срав-

нить e (ординату пересечения отрезка с прямой x = 1) с $\frac{1}{2}$. Далее, для следующего шага алгоритм работает аналогично с учетом изменения e - ординаты пересечения отрезка со следующей вертикальной прямой $x = k, k \in N$.

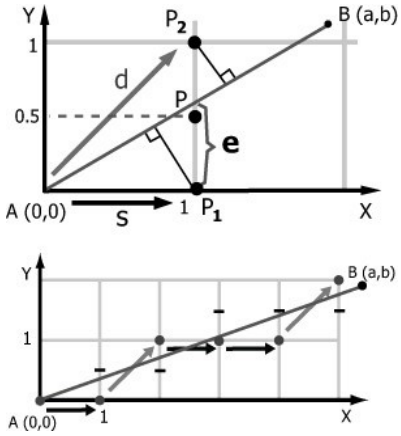


Рис. 4. Цифровой дифференциальный анализатор.

Недостатком данного алгоритма является то, что он работает с числами с плавающей точкой.

Алгоритм Брезенхема

Брезенхем модифицировал алгоритм DDA, чтобы он работал в целых числах.

Алгоритм Брезенхема был создан им для вывода отрезков на цифровых инкрементальных графопостроителях, которые могли осуществлять лишь простые единичные сдвиги печатающей головки. Дальнейшая оптимизация может быть произведена, если заметить, что отрезок симметричен относительно прямой, проходящей перпендикулярно ему через его середину; в этом случае можно начинать рисовать сразу с двух концов, что сократит число итераций цикла в алгоритме вдвое.

Изображение окружностей

Для начала перейдем к канонической системе координат, в которой центр окружности совпадает с началом координат. Тогда можно заметить, что в силу симметрии окружности относительно прямых, разделяющих октанты, достаточно построить растровое представление в одном октанте, а затем с помощью симметрий получить изображения в других октантах (рис.

5). Будем пользоваться заданием окружности в виде неявной функции: $x^2 + y^2 - R^2 = 0$.

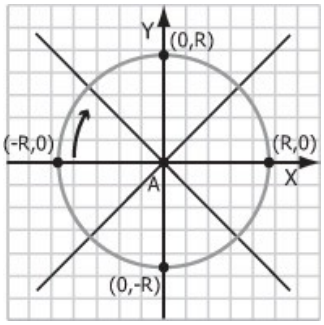


Рис. 5. Симметрии при изображении окружности.

Пусть $f(x, y) = x^2 + y^2 - R^2$. Будем рисовать часть окружности в 4-м октанте, начиная с точки (-R, 0) (рис. 5, показано стрелкой).

Пусть $R \in N$, тогда $f(x,y) \in Z$ для $x \in Z, y \in Z$. Пусть функция plot8(x, y) отображает на растре все 8 точек, полученных из (x, y) с помощью симметрий.

Алгоритм Брезенхема

Будем рассуждать подобно алгоритму Брезенхема для отрезков (с соответствующими поправками на 4-й октант). Из двух возможных пикселов в 4-м октанте (соответствующих вертикальному и диагональному смещениям, которые обозначаются аналогично прежним s и d) будем выбирать тот, расстояние от окружности до которого меньше.

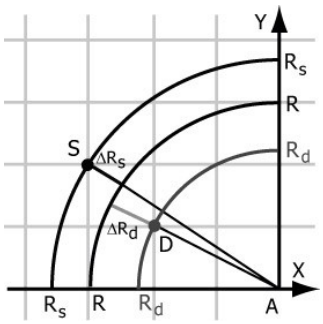


Рис. 6. Расстояния до окружности.

Для того чтобы выбрать один из двух возможных пикселей, будем сравнивать расстояния от них до окружности: где расстояние меньше - тот пиксел и будет искомым. В примере на рис. 6 сравниваются расстояния от

точек $S(x_s, y_s)$ и $D(x_d, y_d)$ до окружности с радиусом R. Из евклидовой метрики получаем:

$$\Delta R_s = \sqrt{x_s^2 + y_s^2} - R;$$
$$\Delta R_d = R - \sqrt{x_d^2 + y_d^2}.$$

Но вычисление квадратного корня - вычислительно трудоемкая операция, поэтому при достаточно больших R мы будем заменять сравнение расстояний сравнением приближенных значений их квадратов (рис. 7):

$$(\Delta R_s)^2 - (\Delta R_d)^2 = x_s^2 + y_s^2 - 2R\sqrt{x_s^2 + y_s^2} + R^2 - x_d^2 - y_d^2 + 2R\sqrt{x_d^2 + y_d^2} - R^2.$$

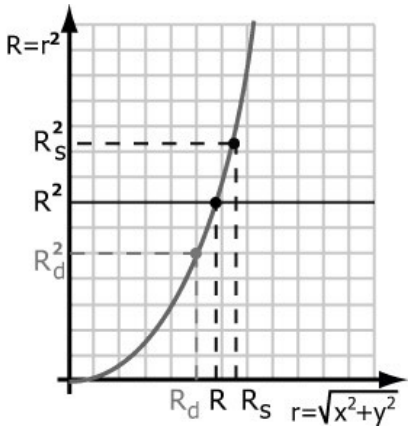


Рис. 7. Приближенное сравнение расстояний.

Уменьшим два слагаемых на приблизительно одинаковые величины:

$-2R\sqrt{x_s^2 + y_s^2}$ заменим на $-2R \cdot R$,

$+2R\sqrt{x_d^2 + y_d^2}$ заменим на $+2\sqrt{x_d^2 + y_d^2} \cdot \sqrt{x_d^2 + y_d^2}$

получим

$$(\Delta R_s)^2 - (\Delta R_d)^2 \approx x_s^2 + y_s^2 - 2R \cdot R + R^2 -$$
$$- x_d^2 - y_d^2 + 2\sqrt{x_d^2 + y_d^2} \cdot \sqrt{x_d^2 + y_d^2} - R^2 =$$
$$= x_s^2 + y_s^2 + x_d^2 + y_d^2 - 2R^2.$$

Таким образом, приближенно

1. D ближе к окружности, чем S $\Leftrightarrow x_s^2 + y_s^2 + x_d^2 + y_d^2 - 2R^2 > 0;$
2. S ближе к окружности, чем D $\Leftrightarrow x_s^2 + y_s^2 + x_d^2 + y_d^2 - 2R^2 < 0.$

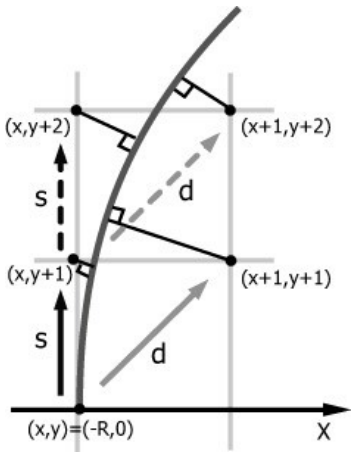


Рис. 8. Алгоритм Брезенхема для окружности

Пусть (x, y) - текущий пиксель. Обозначим

$$F_s = f(x, y+1) = x^2 + (y+1)^2 - R^2 > 0,$$
$$F_d = f(x+1, y+1) = (x+1)^2 + (y+1)^2 - R^2 < 0,$$
$$F = F_s + F_d;$$

$$\Delta F(s) = (\Delta F \text{ при переходе } s\colon (x,y) \to (x, y+1)) =$$
$$= f(x, y+2) + f(x+1, y+2) - f(x, y+1) -$$
$$- f(x+1, y+1)$$
$$= 4y + 6,$$

$$\Delta F(d) = (\Delta F \text{ при переходе } d\colon (x,y) \to (x+1, y+1)) =$$
$$= f(x+1, y+2) + f(x+2, y+2) - f(x, y+1) -$$
$$- f(x+1, y+1)$$
$$= 4x + 4y + 10.$$

Тогда из двух возможных смещений d и s выберем.

1. $F = F_s + F_d < 0 \Leftrightarrow F_s < -F_d$, т.е. (x + 1, y + 1) ближе к окружности, чем (x, y + 1):

d: Переходим в (x + 1, y + 1) и придаем соответствующие приращения F, ΔF(s), ΔF(d):

F = F +ΔF(d); ΔF(s) = ΔF(s)+4; ΔF(d) = ΔF(d)+8.

2. $F = F_s + F_d < 0 \Leftrightarrow F_s < -F_d$, т.е. (x, y+1) ближе к окружности, чем (x + 1, y + 1):

s: Переходим в (x, y + 1) и придаем соответствующие приращения F, ΔF(s), ΔF(d):

F = F +ΔF(s); ΔF(s) = ΔF(s)+4; ΔF(d) = ΔF(d)+4.

Если мы начинаем из (-R, 0), то начальные значения будут следующими:

$$F = F_s + F_d = ((-R)^2 + 1^2 - R^2) + ((-R+1)^2 + 1^2 - R^2)$$
$$= 1 + (-2R + 1 + 1) = 3 - 2R,$$
$$\Delta F(s) = 4 \cdot 0 + 6 = 6,$$
$$\Delta F(d) = 4 \cdot (-R) + 4 \cdot 0 + 10 = 10 - 4R.$$

Легко видеть, что в алгоритме все величины, связанные с F, кроме F начального, будут кратны 2. Но, если мы поделим все эти величины на 2 (в дальнейшем значения всех величин уже понимаются в этом смысле), то $F_{\text{нач}} = \frac{1}{2} + 1 - R$. Так как приращения F могут быть только целочисленными, то $F = \frac{1}{2} + T$, где $T \in Z$; т.е. если отнять $\frac{1}{2}$от всех значений, то знак F не изменится для всех T, кроме T = 0. Для того чтобы результат сравнения остался прежним, будем считать, что F = 0 теперь соответствует смещению s.

```
x = -r; y = 0;
F = 1-r;
ΔFs = 3;
ΔFd = 5-2*r;
while( x + y < 0 )
{
    plot8( x, y );
    if( F > 0 )
    {
        // d: Диагональное смещение
        F += ΔFd;
        x++; y++;
        ΔFs += 2;
        ΔFd += 4;
    }
    else
    {
```

```
            // s: Вертикальное смещение
            F += ΔFs;
            y++;
            ΔFs += 2;
            ΔFd += 2;
        }
    }
```

Листинг 3. Алгоритм Брезенхема для окружности

Размерность вычислений этого алгоритма (т.е. отношение максимальных модулей значений величин, с которыми он оперирует к модулям исходных данных (в данном случае R)) равна 2.

Лекция 8. Буферы кадра (2 часа)

За вывод графической информации на дисплей в ПК отвечает специальный набор микросхем, обычно помещаемый на отдельную плату, которая называется **видеоплатой** или **видеокартой**. Основной задачей является преобразование образа экрана, находящегося в памяти, т.н. **кадрового буфера** (англ. frame buffer), в набор сигналов, понятных дисплею. Для подключения дисплеев к компьютерам используются стандарты, устанавливающие логические и физические параметры соединения. В настоящее время два самых распространенных из них - это аналоговый VGA и цифровой DVI. Первый используется для подключения как аналоговых по сути дисплеев на ЭЛТ, так и цифровыхЖК-дисплеев (аналого-цифровое преобразование в этом случае происходит в самом дисплее), второй - исключительно для ЖК-дисплеев.

На видеокарте присутствует видеопамять, характерной особенностью которой является то, что она двухпортовая - подсоединена как к шине, по которой передаются данные от центрального процессора, так и к микросхеме, отвечающей за вывод на дисплей, и они могут одновременно обращаться к ней. В дешевых устройствах в качестве видеопамяти используется часть основной памяти, что значительно замедляет обработку данных на видеокар-

те. Объем памяти должен быть достаточным для хранения данных кадрового буфера, а желательно еще и **вторичного буфера** (англ. back buffer), если используется технология **двойной буферизации**. Дело в том, что если менять значения пикселей прямо в момент вывода их на экран, то при высокой частоте обновления могут возникать артефакты, связанные с тем, что на экран выводится еще не отрисованное до конца изображение. Чтобы этого избежать, при двойной буферизации во время вывода изображения из области видеопамяти, которая назначена кадровым буфером (называемым в этом случае также **первичным буфером** (англ. front buffer)), изображение следующего кадра строится во вторичном буфере, а при показе следующего кадра эти области памяти меняются ролями. Эта технология используется для показа динамичных изображений, таких как игры. Дополнительная видеопамять также ускоряет обработку графики, позволяя держать дополнительные графические элементы, которые отображаются в кадровом буфере с помощью **блиттинга** (об этом см. ниже).

Доступ к видеопамяти со стороны процессора может быть организован двояко - либо видеопамять, как часть адресов, включается в адресное пространство процессора, либо для копирования данных между основной и видеопамятью контроллеру на видеокарте посылается специальная команда и копирование происходит с помощью DMA (от англ. Direct Memory Access - микросхема, позволяющая осуществлять передачу данных в/из оперативной памяти периферийным устройствам без участия центрального процессора).

Микросхема, отвечающая за вывод на дисплей, постоянно сканирует видеопамять, преобразует ее в форму, соответствующую интерфейсу дисплея, и формирует выходной сигнал, передаваемый по кабелю на дисплей (см. рис. 2.6). Если видеоплата оснащена аналоговым выходом, то в нее должен быть встроен цифро-аналоговый преобразователь, называемый RAMDAC - Random Access Memory Digital to Analog Converter. Так как информация о пикселях передается последовательно, то RAMDAC должен об-

ладать достаточно высокой тактовой частотой, чтобы позволять выводить изображения высокого разрешения с достаточной частотой обновления. Например, изображение 1600×1200 для вывода с частотой 75 Гц требует частоты RAMDAC равной 1600×1200×75 = 144 МГц. Частота работы RAMDAC является важной характеристикой видеокарты.

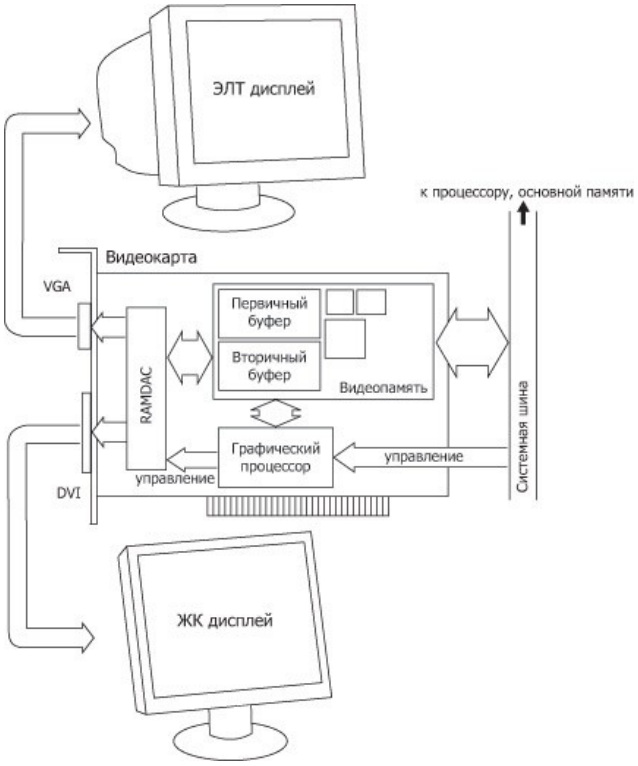


Рис. 1. Архитектура видеоподсистемы ПК.

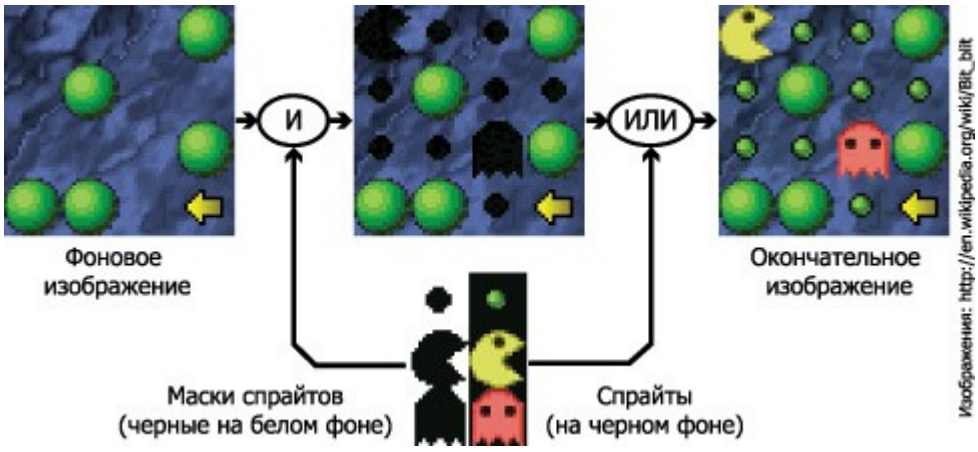


Рис. 2. Пример отрисовки спрайтов.

Также на видеоплате содержится графический процессор, способный быстро выполнять основные операции по работе с изображениями в видеопамяти, которые могут быть разделены на несколько классов.

- **Работа с прямоугольными блоками**. Эта микросхема называется **блиттер**, потому что основная операция которую она производит - это BitBlt (Bit Block Transfer), то есть копирование прямоугольного блока изображения в другое изображение с возможным применением побитовых логических операций (И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ). Это часто используемая операция для помещения объектов произвольной формы, т.н. **спрайтов**, на изображение. Пример того, как это можно сделать с помощью двух BitBlt операций см. на рис. 2. (для понимания необходимо учесть, что пиксели черного цвета состоят из одних 0 битов, а белого - из одних 1 битов). Также видеокарта может поддерживать операцию StretchBlt, - это то же самое, что и BitBlt, но с растяжением по осям.
- **Растеризация примитивов** позволяет производить растеризацию простейших объектов, таких как отрезки, окружности, эллипсы, прямоугольники, многоугольники. Также может поддерживаться заливка одноцветных зон другим цветом или по шаблону. При этом может использоваться и аппаратный **антиалиасинг**. Эти алгоритмы подробно рассмотрены в данной книге. К этой группе также можно отнести аппаратную поддержку отрисовки курсора.
- **Поддержка вывода символов**. Этот блок отвечает за вывод символов на экран определенным шрифтом. Иногда шрифт можно варьировать или загружать из основной памяти свой. Данный блок активно используется, когда видеокарта находится в **текстовом режиме**, когда дисплей логически делится на определенное количество прямоугольных ячеек, чаще всего 80×25, в каждую из которых может быть помещен один символ из ограниченного поднабора ASCII[1)]. Вид каждого из

этих символов определяется в специальной таблице видеокарты, которая может быть изменена. В настоящее время этот режим используется при загрузке ПК, а также при работе в режиме терминала (чаще используется в ОС Linux).

- **Аппаратное ускорение видео и фильтрация изображения**. Кодирование и декодирование видео - очень ресурсоемкая операция, связанная с обработкой больших объемов данных. Некоторые видеокарты способны аппаратно декодировать **видеопоток**, т.е. последовательность сжатых видеоданных, которая соответствует определенному формату. Чаще всего это стандарт MPEG-2, которым закодированы фильмы на DVD. В современных видеокартах также начинает появляться поддержка **Телевидения высокой четкости** (англ. HDTV - High Definition Television). Также возможно и аппаратное масштабирование видео. Аппаратное отображение видео в части экрана носит название **оверлея** (англ. overlay). Некоторые видеокарты также могут аппаратно производить фильтрацию изображений, а также осуществлять гамма-коррекцию .

Практически все современные видеокарты содержат также процессор обработки трехмерной графики, но эта тема выходит за рамки данной книги.

Дополнительно на той же плате часто выведены стандартные разъемы для подключения телевизоров и видеотехники (т.н. TV In/Out) и установлены микросхемы первичной обработки этой информации, в частности, YUV/RGB преобразования.

Некоторые современные видеокарты позволяют подключать сразу несколько дисплеев одновременно.

Лекция 9. Растровая развертка сплошных областей

Одной из уникальных характеристик растрового графического устройства является возможность представления сплошных областей. Генерацию сплошных областей из простых описаний ребер или вершин будем называть растровой разверткой сплошных областей, заполнением многоугольников или заполнением контуров. Для этого можно использовать несколько методов, которые обычно делятся на 2 категории: рстровая развертка и затравочное заполнение.

В методах растровой развертки пытаются определить в порядке сканирования строк, лежит ли точка внутри многоугольника или контура. Эти алгоритмы обычно идут от "верха" многоугольника к "низу".

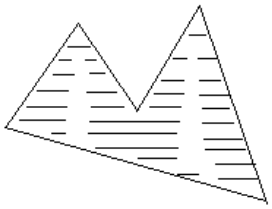


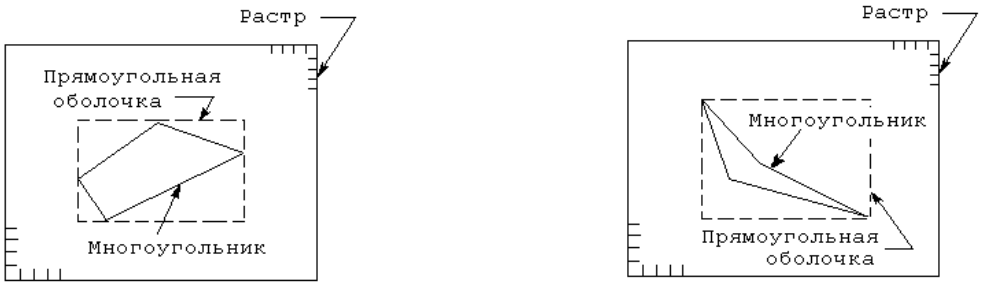
Рис. 1. Штриховка или закраска контура.

В методах затравочного заполненияпредполагается, что известна некоторая точка (затравка) внутри замкнутого контура. В алгоритмах ищут точки, соседние с затравочной и расположенные внутри контура. Если соседняя точка расположена не внутри контура, то значит обнаружена граница контура. Если же точка расположена внутри контура, то она становится затравочной точкой и поиск продолжается рекурсивно. Подобные алгоритмы применимы только к растровым устройствам.

Заполнение многоугольников

Многие замкнутые контуры являются простыми многоугольниками. Ели контур состоит из кривых линий, то его можно аппроксимировать подходящим многоугольником или многоугольниками. Простейший метод заполнения многоугольника состоит в проверке на принадлежность внутренности многоугольника каждого пиксела в растре. Этот методслишком расточителен. Затраты можно уменьшить путем вычисления для многоугольника прямоугольной оболочки - наименьшего многоугольника, содержащего вну-

три себя многоугольник. Как показано на рисунке 2.1. проверяются только внутрение точки этой оболочки. Испльзование прямоугольной оболочки для многоугольника изображенного на рис. 2..(а), намного сокращает число проверяемых пикселей. В тоже время для многоугоьника на рис. 2. (б) число сокращение существенно меньше.



а б

Рис. 2. Прямоугольная оболочка многоугольника.

Растровая развертка многоугольников.

Можно разработать более эффективный метод, чем тест на принадлежность внутренней части, если воспользоваться тем фактом, что соседние пикселы, вероятно, имеют одинаковые характеристики (кроме пикселов граничных ребер). Это свойство называется пространственной когерентностью. Для растровых графических устройств соседние пикселы на сканирующей строке, вероятно, имеют одинаковые характеристики. Это когерентность растровых строк.

Характеристики пикселов на данной строке изменяются только там, где ребро многоугольника пересекает строку. Эти пересечения делят сканирующую строку на области.

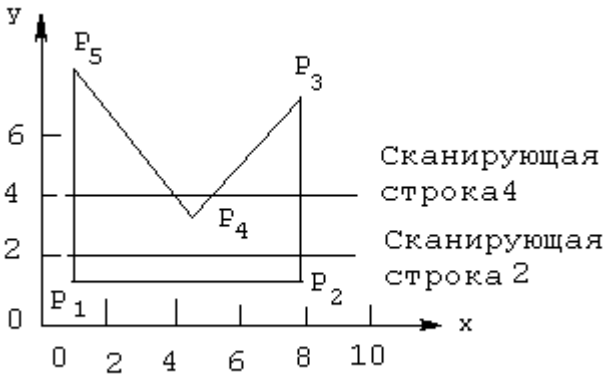


Рис. 3. Растровая развертка сплошной области.

Для простого многоугольника на рис. 3.1 строка 2 пересекает многоугольник при $x = 1$ и $x = 8$. Получаем три области:

x < 1	вне многоугольника
1 <= x <= 8	внутри многоугольника
x > 8	вне многоугольника

Строка 4 делится на пять областей:

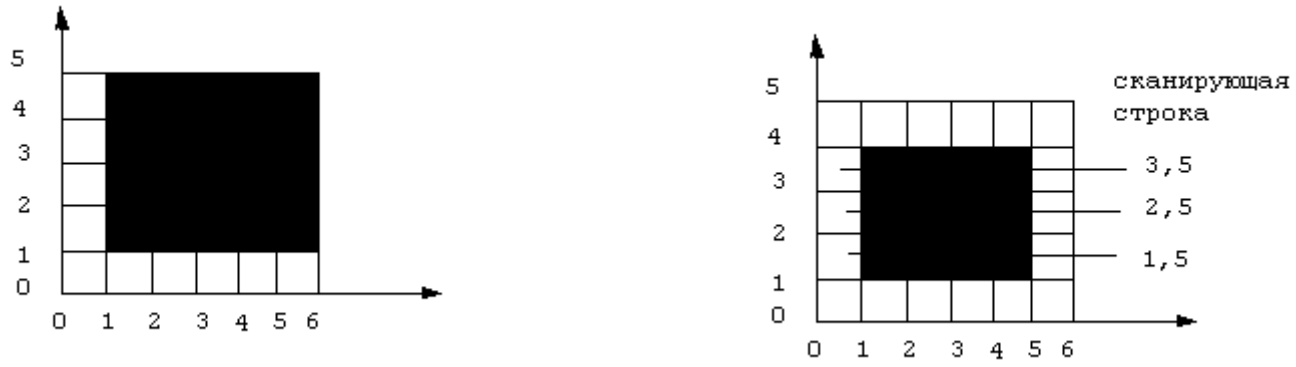
x < 1	вне многоугольника
1<= x <= 4	внутри многоугольника
4 < x < 6	вне многоугольника
6 <= x <= 8	внутри многоугольника
x > 8	вне многоугольника

Совсем необязательно, чтобы точки пересечения для строки 4 сразу определялись в фиксированном порядке (слева направо). Например, если многоугольник задается списком вершин $P_1P_2P_3P_4P_5$, а список ребер - последовательными парами вершин P_1P_2 , P_2P_3, P_3P_4 , P_4P_5 , P_5P_1 то для строки 4 будут найдены следующие точки пересечения с ребрами многоугольника: 8, 6, 4, 1. Эти точки надо отсортировать в возрастающем порядке по *x,* т. е. получить 1, 4, 6, 8.

При определении интенсивности, цвета и оттенка пикселов на сканирующей строке рассматриваются пары отсортированных точек пересечений. Для каждого интервала, задаваемого парой пересечений, используется интенсивность или цвет заполняемого многоугольника. Для интервалов между парами пересечений и крайних (от начала строки до первой точки пересечения и от последней точки пересечения до конца строки) используется фоно-

вая интенсивность или цвет. На рис. 3.1 для строки 4 в фоновый цвет установлены пикселы: от 0 до 1, от 4 до 6, от 8 до 10, тогда как пикселы от 1 до 4 и от 5 до 8 окрашены в цвет многоугольника.

Точное определение тех пикселов, которые должны активироваться. требует некоторой осторожности. Рассмотрим простой прямоугольник, изображенный на рис.4. Прямоугольник имеет координаты (1,1) (5,1), (5,4), (1,4). Сканирующие строки с 1 по 4 имеют пересечения с ребрами многоугольника при $x = 1$ и 5. Вспомним, что пиксел адресуется координатами своего левого нижнего угла, значит, для каждой из этих сканирующих строк будут активированы пикселы с x-координатами 1, 2, 3, 4 и 5. На рис. 4 (а) показан результат. Заметим, что площадь, покрываемая активированными пикселами, равна 20, в то время как настоящая площадь прямоугольника равна 12.



а б

Рис. 4 Система координат строк сканирования.

Модификация системы координат сканирующей строки и теста активации устраняет эту проблему, как это показано на рис. 3.2 (б). Считается, что сканирующие строки проходят через центр строк пикселов, т. е. через середину интервала, как это показано на рис. 4 (б). Тест активации модифицируется следующим образом: проверяется, лежит ли внутри интервала центр пиксела, расположенного справа от пересечения. Однако пикселы все еще адресуются координатами левого нижнего угла. Как показано на рис. 3.2 (б), результат данного метода корректен.

Горизонтальные ребра не могут пересекать сканирующую строку и, таким образом, игнорируются. Это совсем не означает, что их нет на рисунке. Эти ребра формируются верхней и нижней строками пикселов, как показано на рис. 4. Рис. 4 иллюстрирует корректность верхнего и нижнего ребер многоугольника, полученных в результате модификации системы координат сканирующих строк.

Дополнительная трудность возникает при пересечении сканирующей строки и многоугольника точно по вершине, как это показано на рис. 5.

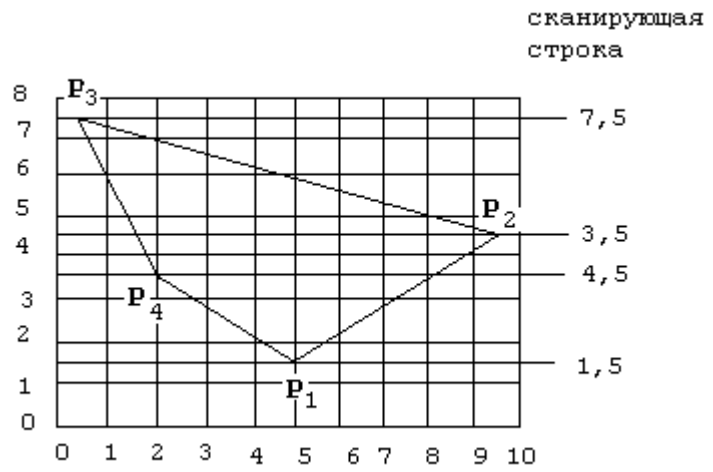


Рис.5. Особенности пересечения со строками сканирования.

При использовании соглашения о середине интервала между сканирующими строками получаем, что строка $y = 3{,}5$ пересечет многоугольник в 2, 2 и 8, т. е. получится нечетное количество пересечений. Следовательно, разбиение пикселов на пары -даст неверный результат, т. е. пикселы (0, 3), (1, 3) и от (3, 3) до (7, 3) будут фоновыми, а пикселы (2, 3), (8, 3), (9, 3) окрасятся в цвет многоугольника. Здесь возникает идея учитывать только одну точку пересечения с вершиной. Тогда для строки $y = 3{,}5$ получим правильный результат. Однако результат применения метода к строке $y = 1{,}5$, имеющей два пересечения в (5, 1), показывает, что метод неверен. Для этой строки именно разбиение на пары даст верный результат, т. е. окрашен будет только пиксел (5, 1), Если же учитывать в вершине только одно пересечение, то пикселы от

(0, 1) до (4, 1) будут фоновыми, а пикселы от (5, 1) до (9, 1) будут окрашены в цвет многоугольника.

Правильный результат можно получить, учитывая точку пересе-чения в вершине два раза, если она является точкой локального ми-нимума или максимума и учитывая ее один раз в противном слу-чае. Определить локаль-ный максимум или минимум многоугольни-ка в рассматриваемой вершине можно с помощью проверки концевых точек двух ребер, соединенных в вер-шине. Если у обоих концов координаты *y* больше, чем у вершины, значит, вершина является точкой локального минимума. Если меньше, значит, вер-шина - точка локального максимума. Если одна больше, а другая меньше, следовательно, вершина не является ни точкой локального минимума, ни точкой локального максимума. На рис. 3.3 точка P_1 - локальный минимум, P_3 - локальный максимум, а P_2,P_4 - ни то и ни другое. Следовательно, в точках P_1 и P_3 учитываются два пересечения со сканирующими строками, а в P_2 и P_4 - одно.

Простой алгоритм с упорядоченным списком ребер.

Используя описанные выше методы, можно разработать эффективные алгоритмы растровой развертки сплошных областей, называемые алгоритма-ми с упорядоченным списком ребер. Они зависят от сортировки в порядке сканирования точек пересечений ребер много-угольника со сканирующими строками. Эффективность этих алгоритмов зависит от эффективности сорти-ровки. Приведем очень простой алгоритм.

Простой алгоритм с упорядоченным списком ребер

Подготовить данные:

Определить для каждого ребра многоугольника точки пересечений со сканирующими строками, проведенными через середины интервалов, для чего можно использовать алгоритм Брезенхема или ЦДА. Горизонтальные ребра игнорируются. Занести каждое пересечение (x, y + 1/2) в список. От-

сортировать список по строкам и по возрастанию x в строке; т. е. (x_1, y_1) предшествует (x_2, y_2), если $y_1 > y_2$ или $y_1 = y_2$ их $x_2 <= x_2$.

Преобразовать эти данные в растровую форму:

Выделить из отсортированного списка пары элементов (x_1, y_1) и (x_2, y_2). Структура списка гарантирует, что $y = y_1 = y_2$ и $x_1 <= x_2$. Активировать на сканирующей строке y пикселы для целых значений x, таких, что $x_1 < x + 1/2 <= x_2$.

Лекция 10. Алгоритмы заполнения

Пусть задан многоугольник $P_1P_2 \ldots P_NP_1$ и требуется растеризовать его вместе с внутренними точками. Будем считать, что процедура отсечения при необходимости была уже произведена и многоугольник целиком помещается в растровом окне. Для удобства каждое ребро многоугольника будем задавать координатами (x_1, y_1) и (x_2, y_2) его концов, так, что $y_2 \geq y_1$. Условимся также отсчитывать на экране координату x слева направо, а y - сверху вниз (таким образом, точка (x_1, y_1) будет верхним концом ребра, а (x_2, y_2) - нижним). Большинство алгоритмов заполнения основано на том факте, что любое горизонтальное сечение контура многоугольника состоит из четного числа точек. Это утверждение неверно в двух случаях (см. рис. 6.1):

- когда секущая прямая содержит горизонтальное ребро;
- когда она содержит вершину, а оба смежных ребра лежат выше (ниже) ее.

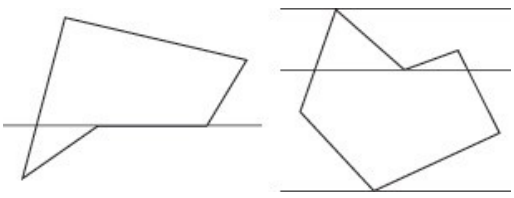


Рис.1. Исключительные случаи.

Однако существуют простые способы исключить эти случаи из рассмотрения; они будут рассмотрены позднее. Пока же будем считать, что приведенное утверждение справедливо всегда. Задача заполнения многоугольника, таким образом, сводится к заполнению определенных промежут-

ков между точками сечения, что должно быть проделано для каждого горизонтального сечения многоугольника. Заметим, что при таком подходе никаких ограничений на свойства многоугольника (выпуклость, отсутствие самопересечений и т.д.) не накладывается.

Алгоритм со списком реберных точек

Этот алгоритм состоит из трех основных этапов.

На первом этапе растеризуются все негоризонтальные ребра многоугольника. Все точки помещаются в списки. Для каждой координаты y_{min}, y_2 . . . y_{max} сопоставим список x-координат всех пикселей, закрашенных при растеризации ребер, которые находятся на горизонтали y (здесь y_{min} и y_{max} - минимальная и максимальная y-координаты пикселей в растровом изображении многоугольника). Формально процедура описывается так:

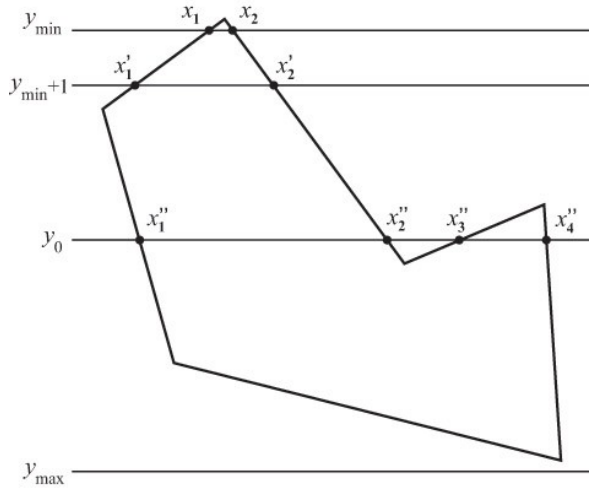


Рис.2. Пример сечений многоугольника.

На втором этапе для каждого y списки упорядочиваются по возрастанию.

На третьем этапе в каждой строке заполняются все отрезки вида $[x_{2i-1}, x_{2i}]$.

Алгоритм со списком активных ребер

Попробуем несколько видоизменить предыдущий алгоритм. Вместо того чтобы хранить в памяти точки пересечения контура с каждой строкой растра, ограничимся лишь одной строкой - текущей. А именно, организуем

список "активных" ребер (CAP), в котором будем хранить информацию обо всех ребрах многоугольника, пересекаемых текущей строкой. Удобство такого подхода в том, что при переходе к новой строке не требуется полностью переформировывать CAP. Достаточно лишь удалить из него "закончившиеся" ребра (то есть ребра из CAP, чей нижний конец оказался выше нового значения y) и добавить вновь появившиеся.

Перейдем к формальному описанию алгоритма. Для каждого ребра создадим структуру данных:

$$y = \lceil y_1 \rceil;$$

$$dx = \frac{x_2 - x_1}{y_2 - y_1};$$

$$x = x_1 + dx \cdot (y - y_1).$$

Все такие структуры поместим в список (далее - y-список) и упорядочим его по возрастанию y.

Ребра, помещенные в CAP, удаляются из y-списка с той целью, чтобы свести проверку наличия в y-списке ребер, начинающихся с данного уровня y, к проверке этого условия для первого ребра в списке (это справедливо в силу упорядоченности списка). Цикл while используется для сохранения упорядоченности CAP, которая может нарушиться при изменении значений x на dx.

Пример такой ситуации показан на рис..3. При ее возникновении указанный цикл выполняет локальную сортировку CAP методом "пузырька".

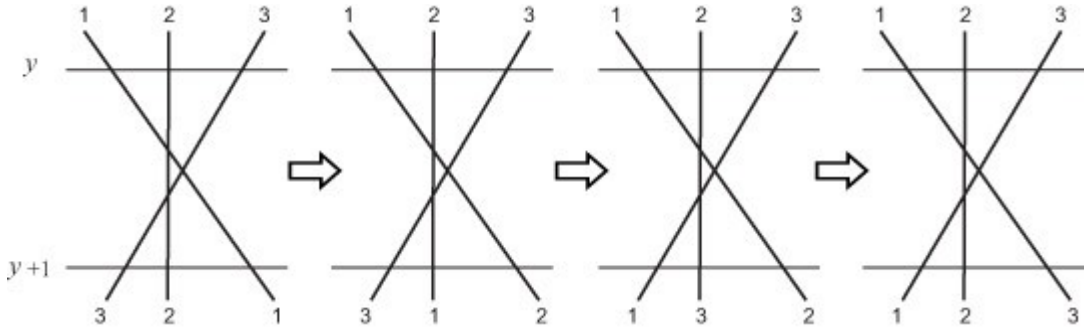


Рис. 3. Локальная сортировка CAP.

Преимущество двух приведенных алгоритмов перед последующими состоит в том, что операции вывода на экран (относительно "медленные" во многих системах) для каждого пикселя выполняются не более одного раза. Недостатком является использование динамических структур данных (списков), что сильно усложняет код и требует дополнительной памяти.

Существует класс систем, в которых использование динамических структур данных нежелательно (вследствие ограниченности ресурса памяти или отсутствия удобных средств разработки программ), в то время как замедление работы из-за частого обращения к видеопамяти некритично (сюда относятся, например, мобильные телефоны и другие портативные устройства, имеющие графический дисплей). В таких системах эффективным будет использование следующих алгоритмов, оперирующих непосредственно с данными в видеобуфере (то есть с "содержимым" экрана).

Заполнение с затравкой

Область, подлежащая заполнению, не всегда задается в виде многоугольника. В этом разделе мы рассмотрим случай, когда заполняемая область задается цветом своей границы. Множество пикселей на растре не задает область однозначно, поэтому требуется задать координаты "затравочного" пикселя, принадлежащего области.

Алгоритмы, рассматриваемые в этом разделе, используют структуру данных под названием **стек**. Стек содержит упорядоченный набор элементов и поддерживает две основные операции: **добавить** элемент и **извлечь** элемент. Вторая операция возвращает элемент, добавленный последним, и удаляет его из набора элементов. Программно стек может быть реализован на основе одномерного массива.

Простейший алгоритм заполнения с затравкой - это так называемый **алгоритм короеда**, получивший подобное название, поскольку заполняемая область последовательно "выедается" по одному пикселю.

При обходе соседних пикселей может рассматриваться и 4-связность (Q принимает 4 значения), и 8-связность (Q принимает 8 значений). В зависимости от этого результат будет различным.

Внутри приведенного алгоритма производится проверка "Q еще не закрашен". Если известно, что пикселей цвета A внутри нашей области изначально не было, то это условие эквивалентно условию "цвет Q != A". В противном случае для проверки этого условия требуется введение специального буфера, где каждому пикселю соответствует флаг закраски, инициализируемый нулем и становящийся единицей при закраске пикселя.

Пример работы алгоритма показан на рис. 6.7. 4-связная область закрашивается в серый цвет и ограничена темными пикселями. Пиксель P извлечен из стека. Он закрашивается серым. Соседние незакрашенные пиксели Q, R и S добавляются в стек.

Используя пространственную когерентность, можно построить более эффективный алгоритм, использующий стек меньшей глубины и закрашивающий за одну итерацию целый горизонтальный отрезок пикселей:

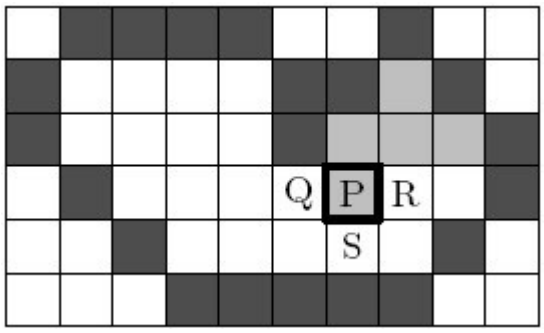


Рис.. Алгоритм короеда.

Пример работы алгоритма показан на рис. . Пиксель P - затравочный; 4-связная область закрашивается в серый цвет и ограничена темными пикселями. При первой итерации производится заполнение целого отрезка. Пиксели Q и R на строке сверху и S на строке снизу добавляются в стек.

Заметим, что приведенные в данном разделе алгоритмы несложно переделать для перекрашивания с затравкой области постоянного цвета A в

другой цвет B. Иными словами, для случая, когда область задается не цветом границы, а собственным цветом. В этом случае алгоритмы даже упрощаются: соседний пиксель добавляется в стек, только если он имеет цвет A (иначе он или не принадлежит перекрашиваемой области, или уже закрашен).

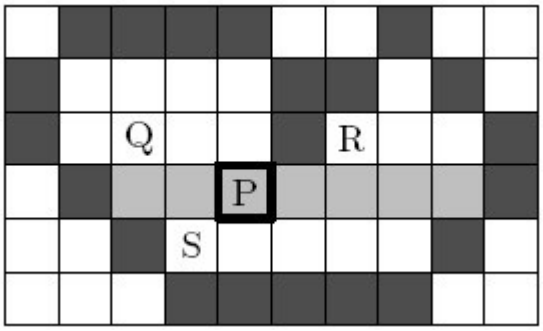
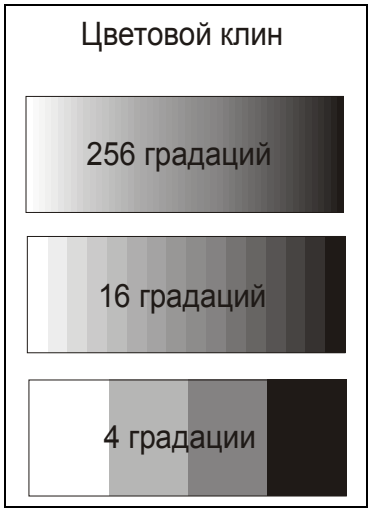


Рис. 6. Заполнение с затравкой по отрезкам.

Лекция 11. Основы методов устранения ступенчатости

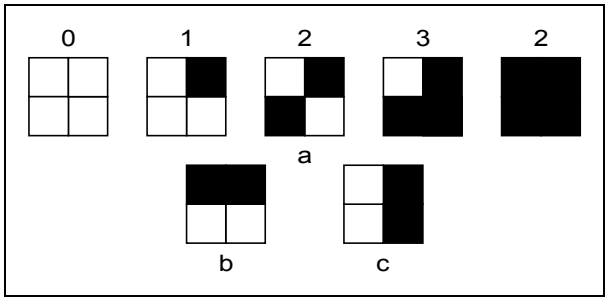
Методы устранения ступенчатости.

Пусть у нас есть только два цвета белый и черный и стоит задача построения цветового клина рис. 2.8.1. 1), т.е. необходимо получить оттенки серого. Для решения этой задачи существуют методы устранения ступенчатости.



Метод полутонов

Сущность: каждый пиксель исходного изображения заменяется группой пикселей .



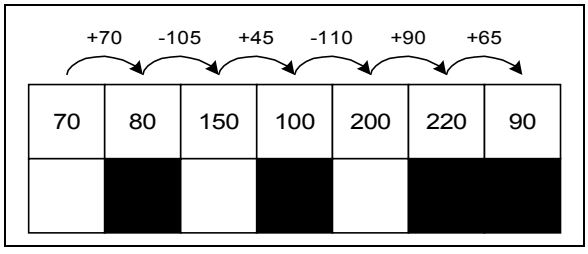
При такой организации получается пять возможных уровней или тонов серого (0-4). В общем случае для двухуровневого дисплея число возможных интенсивностей на единицу больше числа пикселов в клетке. При выборе конфигураций следует проявлять осторожность, так как иначе могут возникнуть нежелательные мелкомасштабные структуры. Использование конфигураций ведет к потере пространственного разрешения, что приемлемо в случае, когда разрешение изображения меньше разрешения дисплея.

Метод переноса.

В данном методе разрешение изображения не изменяется.

Сущность: Предварительно задается число градаций серого, вычисляется середина этого диапазона и, проходя изображение в направлении справа - налево и сверху – вниз, рассматриваются точки изображения : если яркость точки ближе к белому, то ставится белая точка, а если к черному, то ставится черная точка, половина полученной ошибки прибавляется к яркости точки справа, а вторая половина к яркости точки снизу (пропорции можно изменять).

Пример реализации данного алгоритма для прямой приведен на рис.



Метод переноса можно использовать и в случае большего числа градаций.

Хорошие результаты получаются в том случае если начальное значение погрешности для каждой строчки разыгрывается случайным образом, таким образом устраняется регулярность картинки.

Лекция 12. Отсечение

Отсечение, т. е. процесс выделения некоторой части базы данных, играет важную роль в задачах машинной графики.

Отсечение используется и для устранения ступенчатости, помимо своего более привычного применения для отбора той информации, которая необходима для визуализации конкретной сцены или вида, как части более обширной обстановки. Но, кроме того, отсечение применяется в алгоритмах удаления невидимых линий и поверхностей, при построении теней, а также при формировании фактуры. Алгоритмы и понятия, рассматриваемые здесь, можно применить для создания более совершенных алгоритмов, которые отсекают многогранники другими многогранниками. Такие алгоритмы можно использовать для реализации булевых операций, которые нужны в простых системах геометрического моделирования, например при вычислении пересечений и объединении простых тел, ограниченных плоскостями или поверхностями второго порядка. Получающиеся при этом приближенные решения годятся во многих приложениях.

Алгоритмы отсечения бывают

1. Двумерными;
2. Трехмерными.

Эти алгоритмы применяются как к регулярным, так и к нерегулярным областям и объемам. Под регулярными понимаются канонические области и объемы. В частности, к ним относятся прямоугольники и параллелепипеды со сторонами, параллельными осям координат.

Эти алгоритмы можно реализовать:

1. аппаратно;
2. программно.

Алгоритмы отсечения, реализованные программно, зачастую оказываются недостаточно быстродействующими для приложений, ориентированных на процессы, протекающие в реальном времени. Поэтому **трех- и двумерные алгоритмы отсечения реализуются аппаратными или микропрограммными средствами.** В подобных реализациях обычно ограничиваются дву- или трехмерными отсекателями типовых форм. Однако, с появлением сверхбольших интегральных схем (СБИС) открываются возможности для более общих реализации, позволяющих работать в реальном времени как с регулярными, так и с нерегулярными областями и телами.

Двумерное отсечение.

На рис.1 показана плоская сцена и отсекающее окно регулярной формы. Окно задается левым (Л), правым (П), верхним (В) и нижним (Н) двумерными ребрами. Регулярным отсекающим окном является прямоугольник, стороны которого параллельны осям координат объектного пространства или осям координат экрана. **Целью алгоритма отсечения** является определение тех точек, отрезков или их частей, которые лежат внутри отсекающего окна. Эти точки, отрезки или их части остаются для визуализации. А все остальное отбрасывается.

Пример 1. Простое двумерное отсечение

Наклон отрезка от P_1 (- 3/2, 1/6) до P_2 (1/2, 3/2) равен $m = (y_2 - y_1)/ (x_2 - x_1) = (3/2 - 1/6)/ [1/2 - (- 3/2)] = 2/3$. Его пересечения со сторонами окна таковы:

с левой: .x = -1 y = (2/3) [-1 - (- 3/2)] + 1/6 = 1/2

с правой: x =1 y = (2/3)[1 - (- 3/2)] + 1/6 = 11/6

(последнее число больше, чем $y_в$ и поэтому отвергается)

с верхней: y =1 x = - 3/2+ (3/2)[1 - 1/6] = - 1/4

с нижней: y = - 1 x = - 3/2 + (3/2)[- 1 - (1/6)] = - 13/4

(последнее число меньше, чем $x_л$, и поэтому отвергается).

Чтобы разработать схему эффективного алгоритма отсечения, необходимо сначала рассмотреть несколько частных случаев. Напомним, что, как уже указывалось, если наклон бесконечен, то отрезок параллелен левой и правой сторонам окна и надо искать его пересечения только с верхней и нижней сторонами. Аналогично, если наклон равен нулю, то отрезок параллелен верхней и нижней сторонам окна, а искать его пересечения надо только с левой и правой сторонами. Наконец, если код одного из концов отрезка равен нулю, то этот конец лежит внутри окна, и поэтому отрезок может пересечь только одну сторону окна.

Алгоритм отсечения Сазерленда - Коэна, основанный на разбиении отрезка.

Алгоритм, описанный в предыдущем разделе, аналогичен тому, который предложили Коэн и Сазерленд. В алгоритме двумерного отсечения отрезок отсекался поочередно каждой из сторон окна, а для полученных точек пересечения проверялась их принадлежность внутренней области окна, т. е. корректность пересечения. Эта процедура применялась сначала к отрезку P_1P_2 и получался отрезок P'_1P_2, а затем к отрезку P'_1P_2 и получался результирующий отрезок $P'_1P'_2$.

В алгоритме Сазерленда - Коэна отрезок тоже разбивается сторонами окна. **Отличие состоит в том,** что здесь не производятся проверки попадания точки пересечения внутрь окна, вместо этого каждая из пары получающихся частей отрезка сохраняется или отбрасывается в результате анализа кодов ее концевых точек. Рассмотрение отрезка P_1P_2 на рис. 3 показывает трудность реализации этой идеи. Если P_1P_2 разбивается левой стороной окна, то получается два новых отрезка $P_1P'_1$ и P'_1P_2. Коды концевых точек каждого из этих отрезков таковы, что оба они могут быть частично видимы. Следовательно, ни один из них нельзя отвергнуть как невидимый или оставить как

видимый. **Ключом к алгоритму Сазерленда - Коэна** является информацию о том, что одна из концевых точек отрезка лежит вне окна. Поэтому тот отрезок, который заключен между этой точкой и точкой пересечения, можно отвергнуть как невидимый. Фактически это означает замену исходной концевой точки на точку пересечения.

Алгоритм Сазерленда - Коэна формулируется следующим образом:

Для каждой стороны окна выполнить:

Для каждого отрезка P_1P_2 определить, не является ли он полностью видимым или может быть тривиально отвергнут как невидимый.

Если P_1 вне окна, то продолжить выполнение, иначе поменять P_1 и P_2 местами.

Заменить P_1 на точку пересечения P_1P_2 со стороной окна.

Этот алгоритм иллюстрирует следующий пример.

Пример. Алгоритм отсечения Сазерленда - Коэна.

Рассмотрим отсечение отрезка P_1P_2 окном, показанным на рис. 3. Коды концевых точек P_1 (- 3/2, 1/6) и P_2 (1/2, 3/2) равны (0001) и (1000) соответственно. Этот отрезок не является ни полностью видимым, ни тривиально невидимым.

Отрезок пересекает левую сторону окна. P_1 - вне окна.

Пересечение с левой стороны (x = -1) окна происходит в точке P'_1 (-1, 1/2). Замена P_1 на P'_1 дает новый отрезок от P_1 (-1, 1/2) до P_2 (1/2, 3/2).

Коды концевых точек P_1 и P_2 теперь стали (0000) и (1000). Отрезок не является ни полностью видимым, ни тривиально невидимым.

Отрезок не пересекается с правой стороной окна. Перейти к нижней стороне.

Коды концевых точек P_1 и P_2 остаются по-прежнему равными (0000) и (1000). Отрезок не является ни полностью видимым, ни тривиально невидимым.

Отрезок не пересекается с нижней стороной окна. Перейти к верхней.

Коды концевых точек P_1 и P_2 остаются равными (0000) и (1000). Отрезок не является ни полностью видимым, ни тривиально невидимым.

Отрезок не пересекается с верхней стороной окна. P_1 - не снаружи окна. Поменяв P_1 на P_2 местами, получили новый отрезок от P_1 (1/2, 3/2) до P_2 (-1, 1/2).

Точка пересечение с верхней стороной окна (y = -1) равна P'_1 (-1/4, 1). Заменив P_1 на P'_1, получаем новый отрезок от P_1 (-1/4, 1) до P_2 (-1, 1/2).

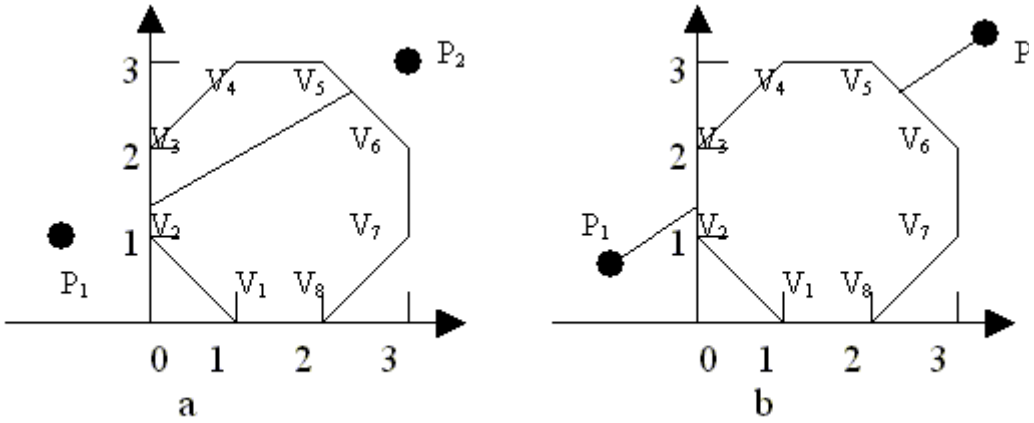
Теперь коды концевых точек P_1 и P_2 равны (0000) и (0000). Отрезок полностью видим.

Процедура завершена.

Начертить отрезок.

Внутреннее и внешнее отсечение.

В предыдущих разделах упор был сделан на отсечение отрезка внутренней областью окна. Однако существует возможность отсечения отрезка и внешней его областью. Для этого надо определить часть или части отрезка, лежащие вне окна, и начертить их. Например, видимые части отрезка P_1 P_2 показанного на рис. 9.4. b, распложены в интервалах 0 <= t < 1/6 и 5/6 < t <= 1, или от точки (- 1, 1) до точки (0, 3/2), а также от точки (7/3, 8/3) до точки (3, 3). Результаты как внутреннего, так и внешнего отсечений этого врезка показаны на рис. 1.



Внешнее отсечение играет важную роль в дисплеях, допускающих работу с несколькими окнами, как это показано на рис. 2, а этом рисунке прио-

ритет окон 1, 2, 3 выше приоритета окна всего экрана, а приоритет окон 1 и 3 выше приоритета окна 2. Поэтому изображение в окне экрана отсекается его собственной областью и внешними областями окон 1, 2, 3. Изображение в 2 отсекается его собственной областью и внешними областями 1 и 3. Изображения в окнах 1 и 3 нужно отсечь только соответствующими внутренними областями.

Внешним отсечением можно воспользоваться и при работе с вогнутым полигональным окном. На рис. 3 показан вогнутый многоугольник, заданный вершинами V_1 V_2 V_3 V_4 V_5 V_6 V_1 . Этот вогнутый многоугольник можно преобразовать в выпуклый путем соединения вершин V_3 и V_5 штриховым отрезком. Отрезок P_1 P_2 внутренне отсекается полученным многоугольником с помощью алгоритма Кируса - Бека. А затем полученный при этом отрезок P'_1 P'_2 внешне отсекается многоугольником V_3 V_4 V_5 V_3, что и дает искомый результат - P'_1 P''_2.

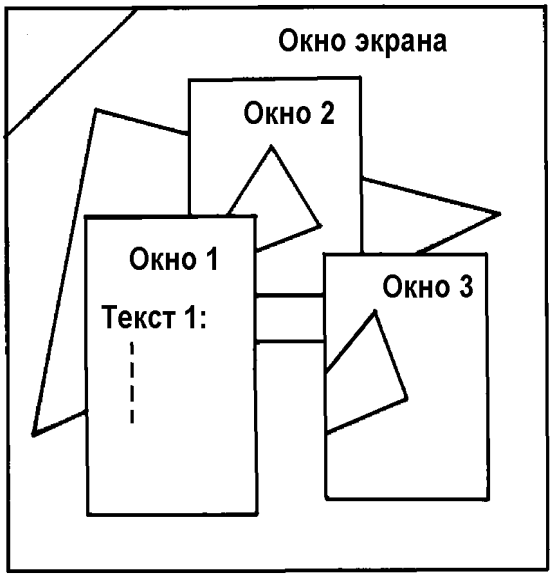


Рис 2. Отсечение несколькими окнами

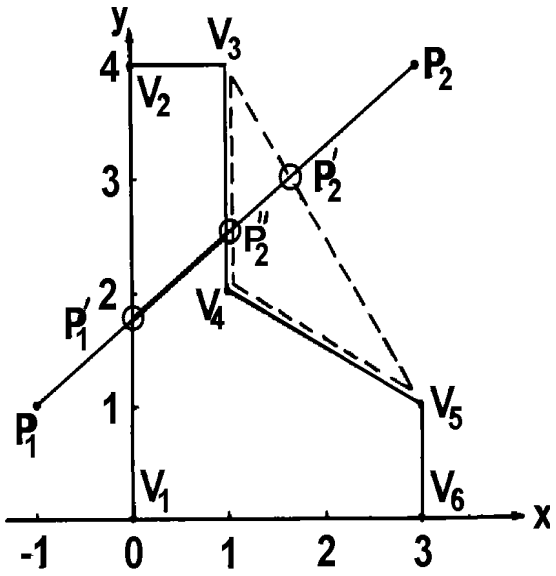


Рис 3 Отсечение невыпуклым окном

Лекция 13. Определение выпуклости трехмерного тела и вычисление внутренних нормалей к его граням

Для работы с алгоритмом Кируса - Бека надо, прежде всего, убедиться, что окно является выпуклым, а затем вычислить внутренние нормали к каждой его стороне. Факт выпуклости или невыпуклости двумерного полигонального окна можно установить путем вычисления векторных произведений его смежных сторон. Выводы, которые можно сделать из анализа знаков этих произведений, таковы:

Все знаки равны нулю - многоугольник вырождается в отрезок.

Есть как положительные,

так и отрицательные знаки - многоугольник невыпуклый.

Все знаки неотрицательные - многоугольник выпуклый, а внутренние нормали ориентированы влево от его контура.

Все знаки неположительны - многоугольник выпуклый, а внутренние нормали ориентированы вправо от его контура.

Рис.1 иллюстрирует эти правила.

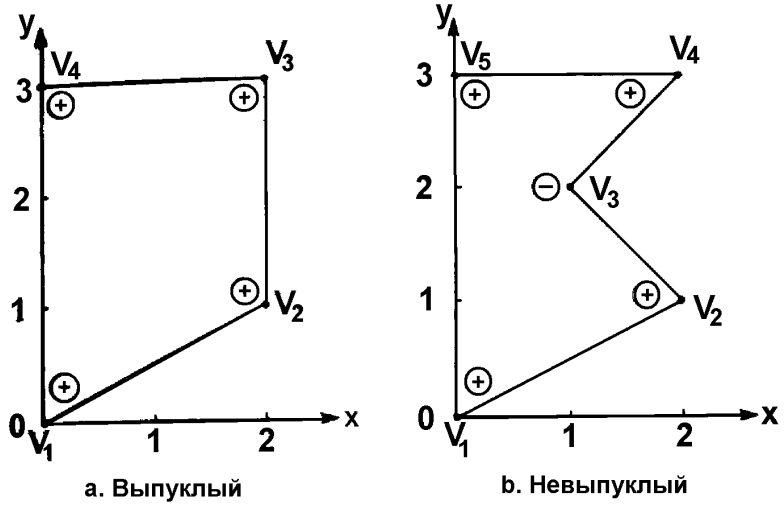


Рис 1. Определение факта выпуклусти многоугольника

Другой подход заключается в том, что одна из вершин многоугольника может быть выбрана базой, и могут вычисляться векторные произведения для пар векторов, начинающихся в этой базе и заканчивающихся в последовательных вершинах многоугольника. Результаты этого метода интерпретируются точно так же.

Векторные произведения будут перпендикулярны к плоскости многоугольника. Векторные произведения двух плоских векторов $\mathbf{V_1}$, и $\mathbf{V_2}$ равно $(Vx_1\ Vy_2 - Vy_1\ Vx_2)$**k,** где k - единичный вектор, перпендикулярный к плоскости, несушей векторы-сомножители.

Нормаль к стороне многоугольника можно вычислить, если вспомнить, что скалярное произведение пары перпендикулярных векторов равно нулю.

Если n_x и n_y - неизвестные компоненты нормали к известному вектору (V_x, V_y) стороны многоугольника, то $n * Vy = (n_x\ i + n_y\ j) * (Ve_x\ i + Ve_y\ j) = n_x\ Ve_x + n_y\ Ve_y = 0$

$n_x\ Ve_x = - n_y\ Ve_y$

Поскольку нас интересует только направление нормали, то положим, что $n_y = 1$ без потери общности. Следовательно, нормаль равна $n= - Ve_y / Ve_x\ i + j$

Если вектор стороны многоугольника образован как разность векторов пары смежных его вершин V_{i-1} и V_i и если скалярное произведение нормали и вектора от V_{i-1} до V_{i+1}. положительно, то n - внутренняя нормаль. В противном случае n - внешняя нормаль. В последнем случае внутреннюю нормаль можно получить, умножив n на -1. Один простой пример проиллюстрирует этот метод.

Пример. Векторное произведение

На рис.1, а показан простой выпуклый многоугольник, а на рис. 1,b - невыпуклый многоугольник. В табл. 9.2 и 9.3 приведены результаты всех вычислений. Рассмотрим, например, векторное произведение сторон, смежных вершине V_2 и внутреннюю нормаль к стороне $V_1\ V_2$ для многоугольника с рис. 1. а.

Стороны, смежные вершине V_2 равны: $V_1\ V_2 = 2i + j$, $V_2\ V_3 = 2\ j$. Векторное их произведение равно $V_1\ V_2$ x $V_2\ V_3 = 4k$, где k - единичный вектор,

перпендикулярный плоскости многоугольника. Это векторное произведение положительно.

Разбиение невыпуклых многоугольников.

Простое обобщение метода поворотов и переносов окна, используемого **для определения факта его выпуклости или невыпуклости** , позволяет разбивать или разделять простой невыпуклый многоугольник на несколько невыпуклых многоугольников. Если вершины многоугольника перечисляются против часовой стрелки, то эта процедура будет иметь вид:

Для каждой i-й вершины многоугольника надо так его перенести, чтобы она совпала с началом координат.

Повернуть многоугольник относительно координат по часовой стрелке так, чтобы (i+1)-я вершина оказалась на положительной полуоси x.

Проанализировав знак ординаты (i + 2)-й вершины. Если он неотрицателен, то многоугольник в (i + 1)-й вершине. Если же этот знак отрицателен, то многоугольник невыпуклый; разбить его.

Многоугольник разрезается вдоль положительной полуоси x, т.е. ищутся такие его стороны, которые пересекаются с осью x. Образуются два новых многоугольника: один состоит из вершин, лежащих выше оси x и ближайшей к началу координат точки пересечения с $x > x_{i+1}$, а второй - из вершин, лежащих ниже оси x и уже упомянутой точки пересечения.

Алгоритм рекурсивно применяется к полученным многоугольникам до тех пор, пока все они не станут выпуклыми.

Этот алгоритм не дает оптимального разбиения в смысле минимального числа выпуклых компонент. Кроме того, алгоритм не сможет корректно разбить многоугольник, стороны которого пересекаются между собой.

Отсечение многоугольников.

Многоугольник можно рассматривать как набор отрезков. В приложениях, связанных с вычерчиванием штриховых изображений, не слишком существенно, если многоугольник разбит на отрезки до его отсечения. Если за-

мкнутый многоугольник отсекается, как набор отрезков, то исходная фигура может превратиться в один или более открытых многоугольников или просто стать совокупностью разрозненных отрезков, как показано на рис. 9.10. Однако, если многоугольники рассматриваются как сплошные области, то необходимо, чтобы замкнутость сохранялась и у результата. Для примера на рис. 2 это означает, что отрезки *bc, ef, fg* и *ha* должны быть добавлены к описанию результирующего многоугольника.

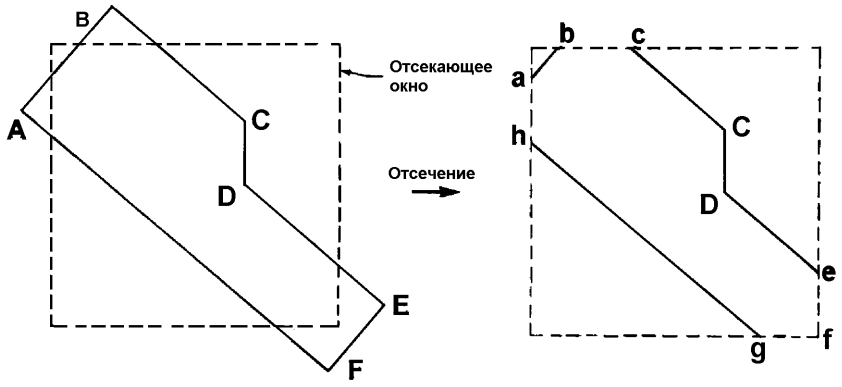


Рис 2. Отсечение многоугольника: открытый многоугольник.

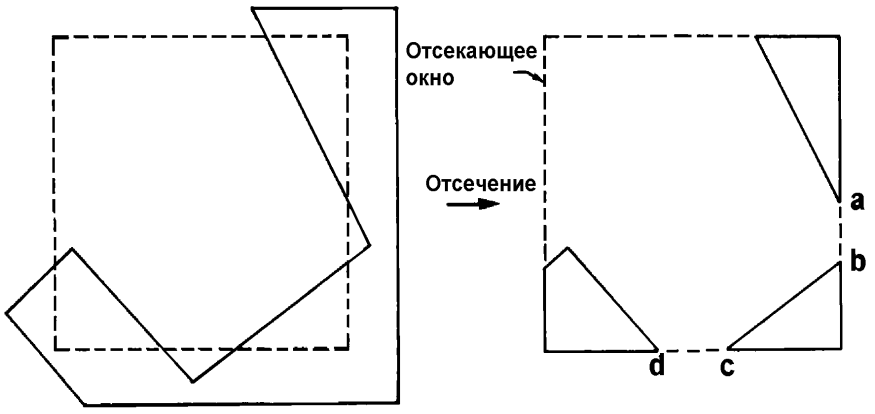


Рис 3. Отсечение многоугольника: не связанные между собой многоугольники.

Добавление отрезков *ef* и *fg* представляет особые трудности. Трудные вопросы возникают и тогда, когда результат отсечения представляет собой несколько несвязанных между собой многоугольников меньших размеров, как это показано на рис. 3. Например, иногда отрезки *ab* и *cd,* показанные на рис. 3, включаются в описание результата. Если, например, исходный много-

угольник объявлен красным на синем фоне, то отрезки *ab* и *cd* тоже будут выглядеть красными на синем фоне. Это противоречит ожидаемому результату.

Последовательное отсечение многоугольника - алгоритм сазерленда - ходжмена

Основная идея алгоритма Сазерленда - Ходжмена состоит в том, что отсечь многоугольник относительно одной прямой или плоскости очень легко. В этом алгоритме исходный и каждый из промежуточных многоугольников отсекается последовательно относительно одной прямой.. Исходный многоугольник задается списком вершин P_1, ..., P_n, который порождает список его ребер $P_1 P_2$, $P_2 P_3$, : ,$P_{n-1} P_n$, $P_n P_1$.

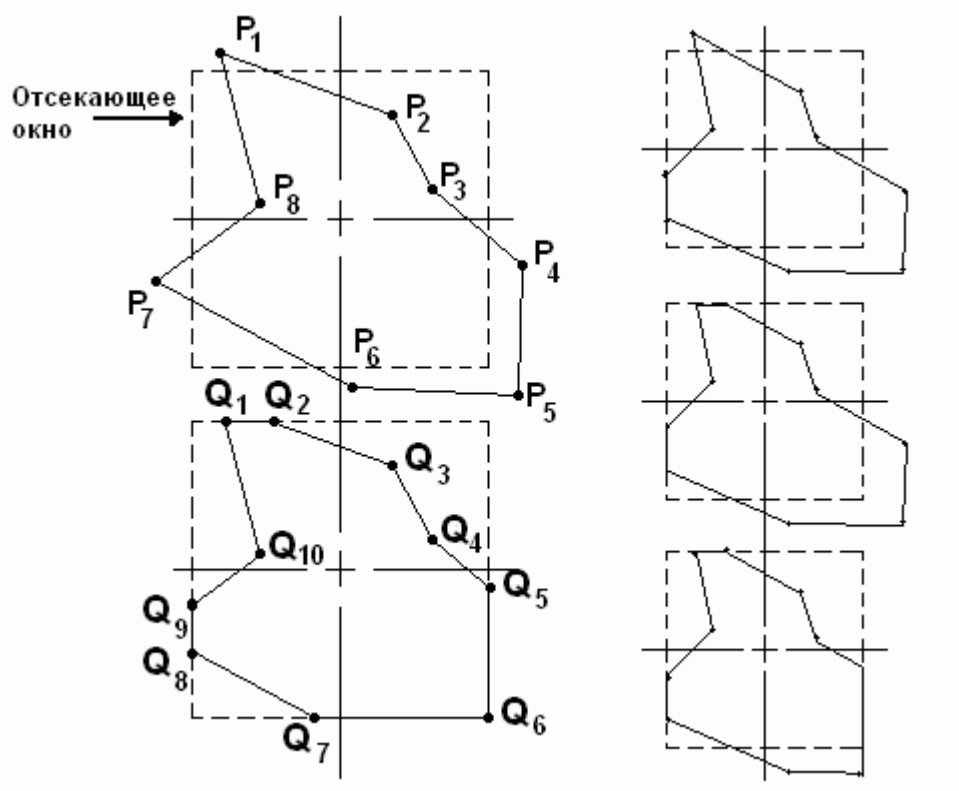


Рис. 4. Последовательное отсечение многоугольника

На рис. 4 показано, что многоугольник сначала отсекается левой стороной окна, в результате чего получается промежуточная фигура. Затем алгоритм вновь отсекает эту фигуру верхней стороной окна. Получается вторая промежуточная фигура. Далее процесс отсечения продолжается с оставшимися сторонами окна. Этапы отсечения показаны на рис. 4. Добавление угло-

вой точки Q_8 в окончательный результат отсечения теперь стало тривиальным. Этот алгоритм способен отсекать любой многоугольник, выпуклый или невыпуклый, плоский или неплоский, относительно любого окна, являющегося выпуклым многоугольником. Порядок отсечения многоугольника разными сторонами окна непринципиален.

Результатом работы алгоритма является список вершин многоугольника, у которого все вершины лежат по видимую сторону от очередной отсекающей плоскости. Поскольку каждая вершина многоугольника отсекается независимо от других, то достаточно рассмотреть только возможные ситуации расположения одного отрезка относительно одной отсекающей плоскости. Будем рассматривать каждую точку P из списка вершин многоугольника, за исключением первой, как конечную точку ребра, начальной точкой S которого является вершина, предшествующая P в этом списке. Тогда возможны только четыре ситуации взаимного расположения ребра и отсекающей плоскости.

Результатом каждого сопоставления ребра многоугольника с отсекающей плоскостью будет занесение в список вершин результирующего усеченного многоугольника нуля, одной или двух вершин.

Если рассматриваемое ребро полностью видимо, то результатом будет вершина P. Заносить в результат начальную вершину S в том случае не надо, так как если вершины рассматриваются поочередно, то S уже была конечной точкой предыдущего ребра и поэтому уже попала в результат. Если же ребро полностью невидимо, то результат не изменяется.

Если ребро видимо не полностью, то оно может входить или выходить из области видимости отсекающей плоскости. Если ребро выходит из области видимости, то надо определить и занести в результат точку пересечения ребра и осекающей плоскости. Если же ребро входит в область видимости, то следует поступить точно так же. Поскольку в последнем случае конечная вершина P ребра видима, то она также должна попасть в результат.

Для первой вершины многоугольника необходимо определить только факт ее видимости. Если вершина видима, то она попадает в результат и становится начальной точкой S. Если же вершина невидима, она тоже становится начальной точкой, но в результат не попадает.

Последнее ребро - P_nP_1 - следует рассмотреть особо. Это реализуется путем запоминания первой вершины многоугольника в F. Тогда последним ребром становится P_nF и его можно обрабатывать точно так же, как и любое другое ребро.

Определение видимости точки эквивалентно определению той стороны границы отсекающей плоскости, по которую лежит эта точка. Если ребра отсекающего многоугольника обходятся по часовой стрелке, то его внутренность лежит по правую сторону от границы. При противоположном порядке обхода она лежит по правую сторону. Методы определения положения (видимости) точки относительно ориентированного отрезка или плоскости:

1. Первый метод сводится к определению знака скалярного произведения вектора нормали на вектор на вектор, начинающийся в произвольной точке на прямой или плоскости и заканчивающийся в пробной точке.
2. Второй метод заключается в подстановке координат пробной точки в уравнение ориентированной прямой или плоскости.
3. Этот метод является вариантом алгоритма Сазерленда и Ходжмена.

Третий метод сводится к проверке знака координаты z у векторного произведения двух векторов, лежащих в одной плоскости. Пусть две точки P_1 и P_2 лежат на отсекающей плоскости, а P_3 - это пробная точка. Эти три точки задают некую плоскость, на которой лежат два вектора P_1P_2 и P_1P_3. Если эту плоскость считать плоскостью xy, то у векторного произведения векторов P_1P_3 x P_1P_2 ненулевой будет компонента z, равная $(x_3 - x_1)(y_2 - y_1) - (y_3 - y_1)(x_2 - x_1)$. Если знак этой компоненты z будет положительным, нулевым или отрицательным, то P_3 будет лежать соответственно справа, на или слева от прямой P_1P_2.

Все эти методы реализуются особенно просто для случая прямоугольных отсекающих окон, стороны которых параллельны координатным осям.

Лекция 14. Удаление невидимых линий и поверхностей

Задача удаления невидимых линий и поверхностей является одной из наиболее сложных в машинной графике. Алгоритмы удаления невидимых линий и поверхностей служат для определения линий ребер, поверхностей или объемов, которые видимы или невидимы для наблюдателя, находящегося в заданной точке пространства.

Сложность задачи удаления невидимых линий и поверхностей привела к появлению большого числа различных способов ее решения. Многие из них ориентированы на специализированные приложения. Наилучшего решения обшей задачи удаления невидимых линий и поверхностей не существует. Для моделирования процессов в реальном времени, например, для авиатренажеров, требуются быстрые алгоритмы, которые могут порождать результаты с частотой видеогенерации (30 кадр/с). Для машинной мультипликации, например, требуются алгоритмы, которые могут генерировать сложные реалистические изображения, в которых представлены тени, прозрачность и фактура, учитывающие эффекты отражения и преломления цвета в мельчайших оттенках. Подобные алгоритмы работают медленно, и зачастую на вычисления требуется несколько минут или даже часов. Строго говоря, учет эффектов прозрачности, фактуры, отражения и т. п. не входит в задачу удаления невидимых линий или поверхностей. Естественнее считать их частью процесса визуализации изображения. Процесс визуализации является интерпретацией или представлением изображения или сцены в реалистической манере. Однако многие из этих эффектов встроены в алгоритмы удаления невидимых поверхностей и поэтому будут затронуты в данной главе. Существует тесная взаимосвязь между скоростью работы алгоритма и детальностью его результата. Ни один из алгоритмов не может достигнуть хороших оценок для

этих двух показателей одновременно. По мере создания все более быстрых алгоритмов можно строить все более детальные изображения. Реальные задачи, однако, всегда будут требовать учета еще большего количества деталей.

Все алгоритмы удаления невидимых линий (поверхностей) включают в себя сортировку. Порядок, в котором производится сортировка координат объектов, вообще говоря, не влияет на эффективность этих алгоритмов. Главная сортировка ведется по геометрическому расстоянию от тела, поверхности, ребра или точки до точки наблюдения. Основная идея, положенная в основу сортировки по расстоянию, заключается в том, что чем дальше расположен объект от точки наблюдения, тем больше вероятность, что он будет полностью или частично заслонен одним из объектов, более близких к точке наблюдения. После определения расстояний или приоритетов по глубине остается провести сортировку по горизонтали и по вертикали, чтобы выяснить, будет ли рассматриваемый объект действительно заслонен объектом, расположенным ближе к точке наблюдения. Эффективность любого алгоритма удаления невидимых линий или поверхностей в большой мере зависит от эффективности процесса сортировки. Для повышения эффективности сортировки используется также когерентность сцены, т.е. тенденция неизменяемости характеристик сцены в малом. В растровой графике использование когерентности для улучшения результатов сортировки в алгоритмах удаления невидимых поверхностей приводит к алгоритмам, которые очень напоминают алгоритмы растровой развертки.

Алгоритмы удаления невидимых линий или поверхностей можно классифицировать по способу выбора системы координат или пространства, в котором они работают. Алгоритмы, работающие в объектном пространстве, имеют дело с физической системой координат, в которой описаны эти объекты. При этом получаются весьма точные результаты, ограниченные, вообще говоря, лишь точностью вычислений. Полученные изображения можно сво-

бодно увеличивать во много раз. Алгоритмы, работающие в объектном пространстве, особенно полезны в тех приложениях, где необходима высокая точность. Алгоритмы же, работающие в пространстве изображения, имеют дело с системой координат того экрана, на котором объекты визуализируются. При этом точность вычислений ограничена разрешающей способностью экрана. Обычно разрешение экрана бывает довольно низким, типичный пример - 512x512 точек. Результаты, полученные в пространстве изображения, а затем увеличенные во много раз, не будут соответствовать исходной сцене. Например, могут не совпасть концы отрезков. Алгоритмы, формирующие список приоритетов, работают попеременно в обеих упомянутых системах координат.

Объем вычислений для любого алгоритма, работающего в объектном пространстве, и сравнивающего каждый объект сцены со всеми остальными объектами этой сцены, растет теоретически как квадрат числа объектов n^2. Аналогично, объем вычислений любого алгоритма, работающего в пространстве изображения и сравнивающего каждый объект сцены с позициями всех пикселов в системе координат экрана, растет теоретически, как nN. Здесь n обозначает количество объектов (тел, плоскостей или ребер) в сцене, а N - число пикселов. Теоретически трудоемкость алгоритмов, работающих в объектном пространстве, меньше трудоемкости алгоритмов, работающих в пространстве изображения, при $n<N$. Поскольку N обычно равно $(512)^2$, то теоретически большинство алгоритмов следует реализовывать в объектном пространстве. Однако на практике это не так. Дело в том, что алгоритмы, работающие в пространстве изображения, более эффективны потому, что для них легче воспользоваться преимуществом когерентности при растровой реализации.

В следующих разделах дается подробное изложение некоторых алгоритмов, работающих как в объектном пространстве, так и в пространстве изображения. Каждый из них иллюстрирует одну или несколько основополагающих идей теории алгоритмов удаления невидимых линий и поверхностей.

Алгоритм плавающего горизонта.

Алгоритм плавающего горизонта чаше всего используется для удаления невидимых линий трехмерного представления функций, описывающих поверхность в виде:

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

Подобные функции возникают во многих приложениях в математике, технике, естественных науках и других дисциплинах.

Предложено много алгоритмов, использующих этот подход. Поскольку в приложениях в основном интересуются описанием поверхности, этот алгоритм обычно работает в пространстве изображения. Главная идея данного метода заключается в сведении трехмерной задачи к двумерной путем пересечения исходной поверхности последовательностью параллельных секущих плоскостей, имеющих постоянные значения координат x, y или z.

Если на текущей плоскости при некотором заданном значении x соответствующее значение y на кривой больше значения y для всех предыдущих кривых при этом значении x, то текущая кривая видима в этой точке; в противном случае она невидима.

Для хранения максимальных значений y при каждом значении x используется массив, длина которого равна числу различимых точек (разрешению) по оси x в пространстве изображения. Значения, хранящиеся в этом массиве, представляют собой текущие значения "горизонта". Поэтому по мере рисования каждой очередной кривой этот горизонт "всплывает". Фактически этот алгоритм удаления невидимых линий работает каждый раз с одной линией.

Алгоритм работает очень хорошо до тех пор, пока какая-нибудь очередная кривая не окажется ниже самой первой из кривых.

Подобные кривые, естественно, видимы и представляют собой нижнюю сторону исходной поверхности, однако алгоритм будет считать их невидимыми. Нижняя сторона поверхности делается видимой, если модифици-

ровать этот алгоритм, включив в него нижний горизонт, который опускается вниз по ходу работы алгоритма. Это реализуется при помощи второго массива, длина которого равна числу различимых точек по оси x в пространстве изображения. Этот массив содержит наименьшие значения y для каждого значения x. Алгоритм теперь становится таким:

Если на текущей плоскости при некотором заданном значении x соответствующее значение y на кривой больше максимума или меньше минимума по y для всех предыдущих кривых при этом x, то текущая кривая видима. В противном случае она невидима.

В изложенном алгоритме предполагается, что значение функции, т.е. y, известно для каждого значения x в пространстве изображения. Однако если для каждого значениях нельзя указать (вычислить) соответствующее ему значение y, то невозможно поддерживать массивы верхнего и нижнего плавающих горизонтов. В таком случае используется линейная интерполяция значений y между известными значениями для того, чтобы заполнить массивы верхнего и нижнего плавающих горизонтов, как показано на рис. 10.6. Если видимость кривой меняется, то метод с такой простой интерполяцией не даст корректного результата. Этот эффект проиллюстрирован рис. 10.7a. Предполагая, что операция по заполнению массивов проводится после проверки видимости, получаем, что при переходе текущей кривой от видимого к невидимому состоянию (сегмент AB на рис. 10.7a), точка (X_{N+K},Y_{N+K}) объявляется невидимой. Тогда участок кривой между точками (X_N,Y_N) и (X_{N+K},Y_{N+K}) не изображается и операция по заполнению массивов не проводится. Образуется зазор между текущей и предыдущей кривыми. Если на участке текущей кривой происходит переход от невидимого состояния к видимому (сегмент CD на рис.10.7a), то точка (X_{M+K},Y_{M+K}) объявляется видимой, а участок кривой между точками (X_M,Y_M) и (X_{M+K},Y_{M+K}) изображается и операция по заполнению массивов проводится. Поэтому изображается и невидимый кусок сегмента CD. Кроме того, массивы плавающих горизонтов не будут содержать

точных значений y. А это может повлечь за собой дополнительные нежелательные эффекты для последующих кривых.

Следовательно, необходимо решать задачу о поиске точек пересечения сегментов текущей и предшествующей кривых.

Существует несколько методов получения точек пересечения кривых. На растровых дисплеях значение координаты x можно увеличивать на 1, начиная с x_n или x_m.Значение y, соответствующее текущему значению координаты x в пространстве изображения, получается путем добавления к значению y, соответствующему предыдущему значению координаты x, вертикального приращения □y вдоль заданной кривой. Затем определяется видимость новой точки с координатами (x+1,y + □y). Если эта точка видима, то активируется связанный с ней пиксел. Если невидима, то пиксел не активируется, а x увеличивается на 1. Этот процесс продолжается до тех пор, пока не встретится x_{n+k} или x_{m+k}. Пересечения для растровых дисплеев определяются изложенным методом с достаточной точностью. Близкий и даже более элегантный метод определения пересечений основан на двоичном поиске

Если на текущей плоскости при некотором заданном значении x соответствующее значение y на кривой больше максимума или меньше минимума по y для всех предыдущих кривых при этом, то текущая кривая видима. В противном случае она невидима. Если на участке от предыдущего (x_n) до текущего (x_{n+k}) значения x видимость кривой изменяется, то вычисляется точка пересечения (x_i). Если на участке от x_n до x_{n+k} сегмент кривой полностью видим, то он изображается целиком; если он стал невидимым, то изображается фрагмент от x_n до x_i; если же он стал видимым, то изображается фрагмент от x_i до x_{n+k}. Заполнить массивы верхнего и нижнего плавающих горизонтов.

Изложенный алгоритм приводит к некоторым дефектам, когда кривая, лежащая в одной из более удаленных от точки наблюдения плоскостей, появляется слева или справа из-под множества кривых, лежащих в плоскостях, которые ближе к указанной точке наблюдения. После обработки кривых n-1

и n верхний горизонт для значений x = 0 и 1, равен начальному значению y; для значений x от 2 до 17 он равен ординатам кривой n; а для значений 18, 19, 20 - ординатам кривой n-1. Нижний горизонт для значений x = 0 и 1 равен начальному значению y; для значений x = 2, 3, 4 - ординатам кривой n; а для значений x от 5 до 20 - ординатам кривой n-1. При обработке текущей кривой (n+1) алгоритм объявляет ее видимой при x = 4. Это показано сплошной линией на рис.10.8. Аналогичный эффект возникает и справа при x = 18. Такой эффект приводит к появлению зазубренных боковых ребер. Проблема с зазубренностью боковых ребер решается включением в массивы верхнего и нижнего горизонтов ординат. Это можно выполнить эффективно, создав ложные боковые ребра. Приведем алгоритм, реализующий эту идею для обеих ребер.

Обработка левого бокового ребра:

Если P_n является первой точкой на первой кривой, то запомним P_n в качестве P_{n-1} и закончим заполнение. В противном случае создадим ребро, соединяющее P_{n-1} и P_n.

Занесем в массивы верхнего и нижнего горизонтов ординаты этого ребра и запомним P_n в качестве P_{n-1}.

Обработка правого бокового ребра:

Если P_n является последней точкой на первой кривой, то запомним P_n в качестве P_{n-1} и закончим заполнение. В противном случае создадим ребро, соединяющее P_n и P_{n-1}.

Занесем в массивы верхнего и нижнего горизонтов ординаты этого ребра и запомним P_n в качестве P_{n-1}.

Теперь полный алгоритм выглядит так:
Для каждой плоскости z = const.

Обработать левое боковое ребро.
Для каждой точки, лежащей на кривой из текущей плоскости:

Если при некотором заданном значении x соответствующее значение y на кривой больше максимума или меньше минимума по y для всех предыдущих кривых при этом x, то кривая видима (в этой точке). В противном случае она невидима.
Если на сегменте от предыдущего (x_n) до текущего (x_{n+k}) значения x видимость кривой изменяется, то вычисляется пересечение (x_i).
Если на участке от x_n до x_{n+k} сегмент кривой полностью видим, то он изображается целиком; если он стал невидимым, то изображается его кусок от x_n до x_i; если же он стал видимым, то изображается его кусок от x_i до x_{n+k}.

Заполнить массивы верхнего и нижнего плавающих горизонтов. Обработать правое боковое ребро.

Если функция содержит очень острые участки (пики), то приведенный алгоритм может дать некорректные результаты. Здесь самая нижняя линия (z=1) содержит пик. При x=8 следующая линия (z=2) объявляется видимой. При x=12 эта линия (z=2) объявляется невидимой, определяется точка пересечения и линия (z=2) изображается от x=8 до этой точки. На участке от x=12 до x=16 эта линия (z=2) вновь становится видимой, определяется новая точка пересечения и кривая изображается от этого пересечения до x=16. Следующая линия (z=3) при x=8 видима; однако она объявляется видимой и при x=12. Следовательно, эта линия изображается на участке от x=8 до x=12, несмотря на то что она заслонена пиком. Этот эффект вызван вычислением значений функции и оценкой ее видимости на участках, меньших, чем разрешающая способность экрана, т.е. тем, что функция задана слишком малым количеством точек. Если встречаются узкие участки, то функцию следует вычислять в большем числе точек. Если в примере на рис.10.9 функцию вычислять в точках с абсциссами 0, 2,4, ... 18, 20, вместо точек 0, 4, ..., 16, 20, то линия z = 3 будет изображена правильно.

Алгоритм плавающего горизонта

Пример работы алгоритма

Рассмотрим геометрические функции, описанные в табл. 1. Эти функции заданы на плоскостях z = 0, 3, 6. В каждой плоскости заданы две линии. Первая из этих линий - прямая, а вторая описывает пилообразную волну, точки которой лежат как выше, так и ниже плоскости, в которой лежит прямая.

Таблица 1.

N Кривой	N точки	x	y	z	Комментарий
1	1	0	0	0	Пилообразная волна
	2	1	4	0	
	3	6	-4	0	
	4	8	0	0	
2	5	0	0	0	Прямая линия
	6	8	0	0	
3	7	0	0	3	Пилообразная волна
	8	2	4	3	
	9	6	-4	3	
	10	8	0	3	
4	11	0	0	3	Прямая линия
	12	0	0	3	
5	13	0	0	3	Пилообразная волна
	14	2	4	6	
	15	6	-4	6	
	16	8	0	6	
6	17	0	0	6	Прямая линия
	18	8	0	6	

Алгоритм плавающего горизонта легко справляется со случаем пары линий, лежаших в одной плоскости z=const. Однако порядок обработки этих линии влияет на конечный результат. Здесь первой будет рассматриваться прямая.

Перед визуализацией к поверхности, описанной в табл. 1, необходимо применить видовое преобразование. Поверхность сначала поворачивается на 30° вокруг оси y, а затем на 15° вокруг оси x. Результат проецируется на плоскость z = 0 из центра проекции, находящейся в бесконечности на оси +z. Матрица размером 4x4 результирующего преобразования однородных координат есть: изменение этого преобразования приводит к результатам, показанным в табл. 2. И результаты отображены на целочисленную сетку в диапазонах 0 <= x <= 100 и 0<= y <= 50, т. е. в координаты пространства изображения. Упорядочивая кривые по приоритету глубины вдоль оси z и вспоминая, что прямые должны быть обработаны (каждая на своей плоскости z = const) в первую очередь. Порядок обработки таков: б, 5, 4, 3, 2, 1.

Алгоритм использующий z буфер.

Это один из простейших алгоритмов удаления невидимых поверхностей. Впервые он был предложен Кэтмулом [8]. Работает этот алгоритм в пространстве изображения. Идея z-буфера является простым обобщением идеи о буфере кадра. Буфер кадра используется для запоминания атрибутов (интенсивности) каждого пиксела в пространстве изображения, z-буфер - это отдельный буфер глубины, используемый для запоминания координаты z или глубины каждого видимого пиксела в пространстве изображения. В процессе работы глубина или значение z каждого нового пиксела, который нужно занести в буфер кадра, сравнивается с глубиной того пиксела, который уже занесен в z-буфер. Если это сравнение показывает, что новый пиксел расположен впереди пиксела, находящегося в буфере кадра, то новый пиксел заносится в этот буфер и, кроме того, производится корректировка z-буфера новым значением z. Если же сравнение дает противоположный результат, то никаких действий не производится. По сути, алгоритм является поиском по x и y наибольшего значения функции z (x, y).

Главное преимущество алгоритма - его простота. Кроме того, этот алгоритм решает задачу об удалении невидимых поверхностей и делает три-

виальной визуализацию пересечений сложных поверхностей. Сцены могут быть любой сложности. Поскольку габариты пространства изображения фиксированы, оценка вычислительной трудоемкости алгоритма не более чем линейна. Поскольку элементы сцены или картинки можно заносить в буфер кадра или в z-буфер в произвольном порядке, их не нужно предварительно сортировать по приоритету глубины. Поэтому экономится вычислительное время, затрачиваемое на сортировку по глубине.

Основной недостаток алгоритма - большой объем требуемой памяти. Если сцена подвергается видовому преобразованию и отсекается до фиксированного диапазона координат z значений, то можно использовать z-буфер с фиксированной точностью. Информацию о глубине нужно обрабатывать с большей точностью, чем координатную информацию на плоскости (x, y); обычно бывает достаточно 20 бит. Буфер кадра размером 512x512x24 бит в комбинации с z-буфером размером 512x512x20 бит требует почти 1.5 мегабайт памяти. Однако снижение цен на память делает экономически оправданным создание специализированных запоминающих устройств для z-буфера и связанной с ним аппаратуры.

Альтернативой созданию специальной памяти для z-буфера является использование для этой цели оперативной или массовой памяти. Уменьшение требуемой памяти достигается разбиением пространства изображения на 4, 16 или больше квадратов или полос. В предельном варианте можно использовать г-буфер размером в одну строку развертки. Для последнего случая имеется интересный алгоритм построчного сканирования. Поскольку каждый элемент сцены обрабатывается много раз, то сегментирование z-буфера, вообще говоря, приводит к увеличению времени, необходимого для обработки сцены. Однако сортировка на плоскости, позволяющая не обрабатывать все многоугольники в каждом из квадратов или полос, может значительно сократить этот рост.

Другой недостаток алгоритма z-буфера состоит в трудоемкости и высокой стоимости устранения лестничного эффекта, а также реализации эффектов прозрачности и просвечивания. Поскольку алгоритм заносит пикселы в буфер кадра в произвольном порядке, то нелегко получить информацию, необходимую для методов устранения лестничного эффекта, основывающихся на предварительной фильтрации. При реализации эффектов прозрачности и просвечивания, пикселы могут заноситься в буфер кадра в некорректном порядке, что ведет к локальным ошибкам.

Хотя реализация методов устранения лестничного эффекта, основывающихся на префильтрации, в принципе возможна [7], практически это сделать трудно. Однако относительно легко реализуются методы постфильтрации (усреднение подпикселов). Напомним, что в методах устранения лестничного эффекта, основывающихся на постфильтрации, сцена вычисляется в таком пространстве изображения, разрешающая способность которого выше, чем разрешающая способность экрана. Поэтому возможны два подхода к устранению лестничного эффекта на основе постфильтрации. В первом используется буфер кадра, заданный в пространстве изображения, разрешение которого выше, чем у экрана, и z-буфер, разрешение которого совпадает с разрешением экрана. Глубина изображения вычисляется только в центре той группы подпикселов, которая усредняется. Если для имитации расстояния от наблюдателя используется масштабирование интенсивности, то этот метод может оказаться неадекватным.

Во втором методе оба буфера, заданные в пространстве изображения, имеют повышенную разрешающую способность. При визуализации изображения как пикселная информация, так и глубина усредняются. В этом методе требуются очень большие объемы памяти. Например, изображение размером 512x512x24 бита, использующее z-буфер размером 20 бит на пиксел, разрешение которого повышено в 2 раза по осям x и y и на котором устранена сту-

пенчатость методом равномерного усреднения, требует почти 6 мегабайт памяти. Более формальное описание алгоритма z-буфера таково:

Алгоритм использующий список приоритетов.

При реализации всех обсуждавшихся алгоритмов удаления невидимых линий и поверхностей устанавливались приоритеты, т. е. глубины объектов сцены или их расстояния от точки наблюдения. Алгоритмы, использующие список приоритетов, пытаются получить преимущество посредством предварительной сортировки по глубине или приоритету. Цель такой сортировки состоит в том, чтобы получить окончательный список элементов сцены, упорядоченных по приоритету глубины, основанному на расстоянии от точки наблюдения. Если такой список окончателен, то никакие два элемента не будут взаимно перекрывать друг друга. Тогда можно записать все элементы в буфер кадра поочередно, начиная с элемента, наиболее удаленного от точки наблюдения. Более близкие к наблюдателю элементы будут затирать информацию о более далеких элементах в буфере кадра. Поэтому задача об удалении невидимых поверхностей решается тривиально. Эффекты прозрачности можно включить в состав алгоритма путем не полной, а частичной корректировки содержимого буфера кадра с учетом атрибутов прозрачных элементов.

Пример работы алгоритма, использующего кластерные приоритеты

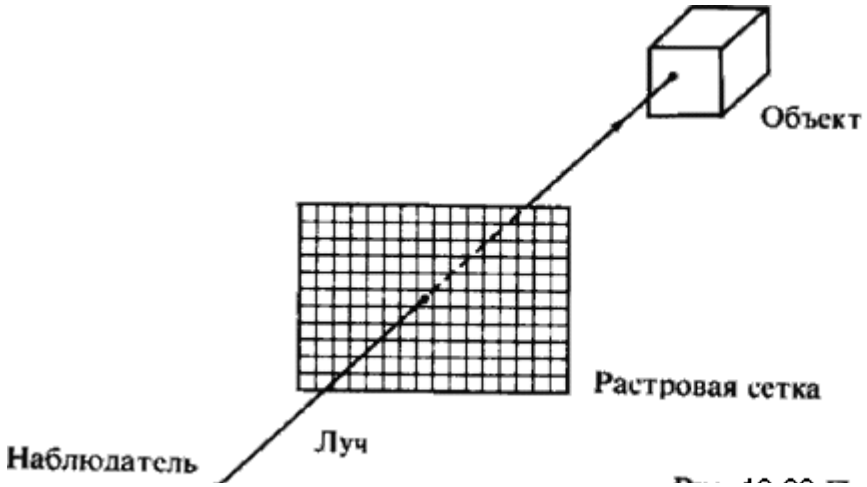
Поскольку подобно алгоритмам Варнока и z -буфера в алгоритмах, строящих список приоритетов, многоугольники обрабатываются в произвольном порядке, применение методов устранения лестничного эффекта к результирующему изображению затруднено. Однако для этой цели здесь применим метод постфильтрации, используемый также в алгоритмах Варнока и z-буфера.

Алгоритмы, строящие список приоритетов, использующие z-буфер и относящиеся к типу Варнока, могут также применяться и для удаления невидимых линий. При их использовании в этом качестве ребра каждого многоугольника заносятся в буфер кадра с одним атрибутом, при чем внутренняя

область каждого многоугольника заносится в буфер кадра с атрибутом фона. При таком подходе многоугольники, которые находятся ближе к точке наблюдения, "заслоняют" ребра многоугольников, которые расположены дальше от нее

Алгоритм использующий трассировку лучей.

Оценка эффективности всех алгоритмов удаления невидимых поверхностей, обсуждавшихся в предыдущих разделах, зависит от определенных характеристик когерентности той сцены, для которой ведется поиск ее видимых участков.



В отличии от них трассировка лучей является методом грубой силы (методом грубой силы принято называть, метод, не учитывающий специфику обрабатываемого объекта). Главная идея, лежащая в основе этого метода, заключается в том, что наблюдатель видит любой объект посредством испускаемого неким источником света, который падает на этот объект и затем каким-то путем доходит до наблюдателя. Свет может достичь наблюдателя, отразившись от поверхности, преломившись или пройдя через нее. Если проследить за лучами света, выпушенными источником, то можно убедиться, что весьма не многие дойдут до наблюдателя. Следовательно, этот процесс был бы вычислительно неэффективен. Аппель первым предложил отслеживать (трассировать) лучи в обратном направлении, т.е. от наблюдателя к объекту, как показано на рисунке 10.20. Это метод был с успехом реализован

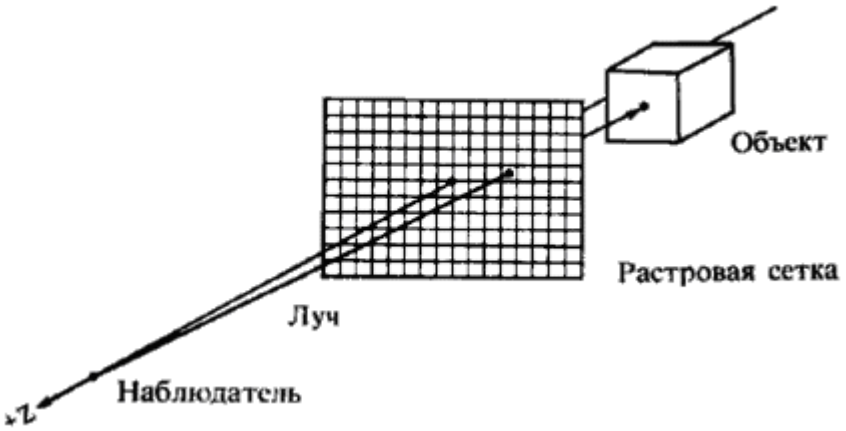
в рамках дисплейной системы визуализации твердых тел MAGI[16]. В самой системе MAGI трассировка прекращалась, как только луч пересекал поверхность видимого непрозрачного объекта; т.е. луч использовался для обработки скрытых или видимых поверхностей. В последствии Кей и Уиттед реализовали алгоритмы трассировки лучей с использованием общих моделей освещения. Эти алгоритмы учитывают эффекты отражения одного объекта от поверхности другого, преломления, прозрачности и затемнения. Производиться также устранение ступенчатости. Настоящее же обсуждение ограничено применениемметода трассировки лучей для определения видимых или скрытых поверхностей.

Рис служит иллюстрацией алгоритма трассировки лучей. В этом алгоритме предполагается, что сцена уже преобразована в пространство изображения. Перспективное преобразование не используется. Считается, что точка зрения или наблюдатель находится в бесконечности на положительной полуоси z. Поэтому все световые лучи параллельны оси z. Каждый луч, исходящий от наблюдателя, походит через центр пиксела на растре до сцены. Траектория каждого луча отслеживается, чтобы определить, какие именно объекты сцены, если таковые существуют, пересекаются с данным лучом. Необходимо проверить пересечение каждого объекта сцены с каждым лучом. Если луч пересекает объект, то определяются все возможные точки пересечения луча и объекта. Можно получить большое количество пересечений, если рассматривать много объектов. Эти пересечения упорядочиваются по глубине. Пересечение с максимальным значением z представляет видимую поверхность для данного пиксела. Атрибуты этого объекта используются для определения характеристик пиксела.

Если точка зрения находится не в бесконечности, алгоритм трассировки луче лишь незначительно усложняется. Здесь предполагается, что наблюдатель по-прежнему находится на положительно полуоси z. Картинная плос-

кость, т.е. растр, перпендикулярна оси z. Задача состоит в том, чтобы построить одноточечную центральную проекцию на картинную плоскость[1-1].

Наиболее важным элементом алгоритма определения видимых поверхностей путем трассировки луче, является процедура определения пересечений. В состав сцены можно включать любой объект, для которого можно создать процедуру построения пересечений. Объекты сцены могут состоять из набора плоских многоугольников, многогранников или тел, ограниченных или определяемых квадратичными или биполиномиальными параметрическими поверхностями. Поскольку 75-96% времени, затрачеваемого алгоритмом трассировки лучей, уходит на определения пересечений, то эффективность процедуры поиска пересечений оказывает значительное влияние на производительность всего алгоритма. Вычислительная стоимость определения пересечений произвольной пространственной прямой (луча) с одним выделенным объектом может оказываться высокой (см. например [21]). Чтобы избавиться от ненужного поиска пересечений, производится проверка пересечения луча с объемной оболочкой рассматриваемого объекта. И если луч не пересекает оболочки, то не нужно больше искать пересечений этого объекта с лучом. В качестве оболочки можно использовать прямоугольный параллелепипед или сферу.



Хотя, как показано, на рис., использование сферы в качестве оболочки может оказаться неэффективным, факт пересечения трехмерного луча со сферой

определяется очень просто. В частности, если расстояние от центра сферической оболочки до луча превосходит радиус этой сферы, то луч не пересекает оболочки. Следовательно, он не может пересечься и с объектом.

ЛАБОРАТОРНЫЕ РАБОТЫ (28 часов)

Лабораторная работа №1 (2-х часовая)
Тема: Запуск и настройка графического редактора AutoCAD 14. Вычерчивание линий. Полилинии и полосы. Многоугольники. Масштабирование объекта рисунка.

Лабораторная работа №2 (4-х часовая)
Тема: Вставка текста в рисунок. Вращение объектов. Зеркальное отображение. Управление слоями. Работа с разными типами линий. Рисунки в качестве блоков. Предопределение блоков. Вставка массива блоков. Трехмерные многоугольные сети.

Лабораторная работа №3 (4-х часовая)
Тема: Рисунки в качестве блоков. Предопределение блоков. Вставка массива блоков. Трехмерные многоугольные сети. Язык AutoLISP как средство адаптации AutoCAD к различным применениям.

Лабораторная работа №4 (4-х часовая)
Тема: Построение линии и окружности по алгоритму простого ЦДА и алгоритму Брезенхема.

Лабораторная работа №5 (2-х часовая)
Тема: Развертка изображения и сплошной области.

Лабораторная работа №6 (2-х часовая)
Тема:Разработка программы реализующей простой гранично-заполняющий алгоритм с затравкой для заполнения внутренней части многоугольника.

Лабораторная работа №7 (2-х часовая)
Тема: Отсечение отрезков по двумерному прямоугольному окну с помощью простого алгоритма и алгоритма Сазерленда – Коэна.

Лабораторная работа №8 (4-х часовая)
Тема: Написать программу, используя метод плавающего горизонта, удаления невидимых линий для поверхности, описанной функцией:

$$F(x,z) = 8\cos(1.2R)/(R+1)$$

$$R = \sqrt{x^2 + z^2}$$

$$-2\pi \le x,z \le 2\pi$$

САМОСТОЯТЕЛЬНАЯ РАБОТА СТУДЕНТОВ (28 часа)

1. Знакомство с публикациями в периодических журналах:
   - Информатика-машиностроение (3час.);
   - Компьютер Пресс (3час.);
   - Автоматизация проектирования (3час.);
   - САПР и графика (3час.);
   - PC WEEK/RE (3час.);
2. Изучение содержания основных разделов книги: Шикин Е.В., Боресков А.В. Компьютерная графика. -М.: "ДИАЛОГ-МИФИ", 1996. (13 час.)

ВОПРОСЫ К ЗАЧЕТУ

1. Координатная и растровая графика. Основные понятия.
2. Распознавание образов по адаптивным эталонам в пирамидальной структуре.
3. Графическая рабочая станция.
4. Распознавание видеообразов.
5. Адресация растра.
6. Основы методов устранения ступенчатости.
7. Групповое кодирование, как способ получения растровой развертки.
8. Двумерное отсечение
9. Буферы кадра.
10. Простой метод устранения лестничного эффекта.
11. Системы координат и преобразования.
12. Распознавание образов по адаптивным эталонам.
13. Растровая развертка сплошных областей. Растровая развертка многоугольников.
14. Алгоритм заполнения по ребрам.
15. Растровые графические дисплеи с регенерацией изображения.
16. Алгоритм плавающего горизонта.
17. Алгоритм Брезенхема построения отрезка.
18. Отсечение многоугольников.
19. Алгоритмы вычерчивания отрезков.
20. Цифровой дифференциальный анализатор.
21. Последовательное отсечение многоугольников.
22. Алгоритм Сазерленда - Ходжмена.
23. Типы графических устройств.
24. Алгоритм Варнока.

25. Восстановление формы скрытых объектов в медицинских исследованиях.
26. Изображение пересекающихся отрезков. Изображение литер.
27. Алгоритмы заполнения с затравкой.
28. Растровая развертка, как способ генерации изображения. Растровая развертка в реальном времени.
29. Алгоритм разбиения средней точкой.
30. Примитивы вывода и атрибуты.
31. Распознавание образов в условиях неизвестной ориентации их корпусов.
32. Устройство ЭЛТ.
33. Удаление невидимых линий и поверхностей.
34. Определение понятия "машинная графика".
35. Простой алгоритм с упорядоченным списком ребер.
36. Графические метафайлы. Диалоговые устройства.
37. Трехмерное отсечение.
38. Клеточное кодирование.
39. Отсечение. Двумерное отсечение.

Критерии оценки

Зачет – Итоговая аттестация по дисциплине. Оценка (зачет) по этим видам контроля складывается с текущей работы студента в семестре, промежуточного контроля, самостоятельной работы и ответе на экзамене (зачете) (40% - промежуточный контроль знаний студентов, 60% - результаты тиового зачета (экзамена)).

Кафедра имеет право перераспределить это соотношение до 10%.

Промежуточный контроль – осуществляется два раза в семестр в виде контрольных точек. Преподаватель проверяет знания студентов в виде контрольных работ, тестов и др. по блоку изученной дисциплины. Фиксируется в журналах успеваемости, находящихся в деканатах.

Результаты учитываются при допуске к сдаче зачета или экзамена.

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

Основная

1. Роджерс Д., Адамс Дж. Математические основы машинной графики. –М.: Мир, 2001. 604 с.
2. Хилл Ф. Программирование компьютерной графики. Для профессионалов. –СПб.: Петер, 2002, 1088 с.
3. Юнь Фень, Программирование графики для Windowsю –СПб.: Петер, 2002, 1088 с.
4. Абламейко С.В., Лагуновски1 Д.М. Обработка изображений: технология, методы, применение. Мн.: Молфея, 200 с.
5. Эйнджел Э., Интерактивная компьютерная графика. Вводный курс на базе Open GL, II изд. Пер с англ. –М.: Изд. Дом «Вильямс», 2001.–532 с.
6. Шикин Е.В., Боресков А.В. Компьютерные модели. –М.: ДИАЛОГ – МИФА, 2000, -464 с.

Дополнительная

1. Гройс Д. Графические средства персонального компьютера. М. Мир. 1989 г.
2. Райн Д. Инженерная графика в САПР. М. Мир. 1983 г.
3. Алгоритмические основы машинной графики. М. Мир. 1989 г.
4. Аугер В. AutoCAD 11.0. Киев., T-UB BAV – 1993г
5. Гладков С.А. Программирование на языке AutoLISP в системе САПР AutoCAD. М., Диалог – «МИФИ», 1991г.