

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
К ЛАБОРАТОРНЫМ РАБОТАМ ПО ДИСЦИПЛИНЕ
«КОМПЬЮТЕРНАЯ ГРАФИКА»
для студентов специальности
6.050101 «Информационно управляющие системы и технологии» (ИУС)

Рассмотрено на заседании кафедры
«Автоматизированные системы
управления»
Протокол № 17 от 18.04.2011 г.

Утверждено на заседании
учебно – издательского совета ДонНТУ
Протокол № 3 от 05.05.2011 г.

Донецк 2011

Методические указания к лабораторным работам по дисциплине «Компьютерная графика» для студентов специальности 6.050101 «Информационно управляющие системы и технологии» (ИУС) / Составили: Васяева Т.А., Теплова О.В. – Донецк: ДонНТУ, 2011. – 80с.

Методические указания содержат краткие теоретические сведения, методические рекомендации и задания для выполнения лабораторных работ по дисциплине «Компьютерная графика». Изложена методика выполнения каждой из лабораторных работ, требования к содержанию отчетов, список рекомендуемой литературы.

Утверждено методической комиссией специальности 6.050101 "Информационно управляющие системы и технологии" (ИУС)

Составители: к.т.н., доц. каф. АСУ Васяева Т.А.
ас. каф. АСУ Теплова О.В.

Ответственный за выпуск: д.т.н., проф., зав. каф. АСУ Скобцов Ю.А.

СОДЕРЖАНИЕ

Введение.....	4
Лабораторная работа № 1.....	5
Лабораторная работа № 2.....	11
Лабораторная работа № 3.....	20
Лабораторная работа № 4.....	32
Лабораторная работа № 5.....	43
Лабораторная работа № 6.....	48
Лабораторная работа № 7.....	55
Лабораторная работа № 8.....	64
Список литературы.....	68
Приложение А.....	69
Приложение Б.....	70
Приложение В.....	71
Приложение Д.....	74
Приложение Е.....	75
Приложение Ж.....	76
Приложение И.....	79

ВВЕДЕНИЕ

Компьютерная графика представляет собой одно из направлений развития систем обработки информации связанной с изображением объектов, которое возникло в связи с необходимостью широкого использования систем компьютерной графики в виртуальной реальности, в глобальной сети Internet и системах интерактивной графики. Понятие «компьютерная графика» очень часто трактуется по-разному. Компьютерная графика – это новая отрасль знаний, которая, с одной стороны, представляет комплекс аппаратных и программных средств, используемых для формирования, преобразования и выдачи информации в визуальной форме на средства отображения ЭВМ. С другой стороны, под компьютерной графикой понимают совокупность методов и приемов для преобразования при помощи ЭВМ данных в графическое представление. Системы компьютерной графики обеспечивают пользователю широкий набор услуг и позволяют создавать целый ряд различных способов диалога, типа человек – компьютер, позволяют создавать анимационные и реалистичные изображения и совершенствуют способы ввода-вывода информации.

Изучение данной дисциплины вносит необходимый вклад в достижение ожидаемых результатов в профессиональной части программы подготовки студентов специальности «Информационно управляющие системы и технологии».

Курс основан на изучении математических основ предметной области, значительное внимание уделяется программной реализации.

Основная задача курса – дать в доступной форме математический аппарат для создания изображений геометрических объектов на экране и их манипуляций, алгоритмы растровой графики; представление пространственных форм, включая алгоритмы удаления скрытых линий и поверхностей.

Лабораторная работа № 1

Тема: построение графических фигур на плоскости

Программирование в графическом режиме

В графическом режиме экран компьютера делится на пиксели; каждый пиксель представляет собой отображение на экране одной точки. Число пикселей на экране (т.е. его разрешающая способность) зависит от типа подключенного к вашей системе видеоадаптера и режима, в который установлен этот адаптер. Для получения на экране графических изображений язык C/C++ предоставляет библиотеку графических функций: можно создавать на экране линии и формы, заполненные шаблонами замкнутые области, а также управлять цветом каждого пикселя.

Язык C/C++ имеет отдельную библиотеку с более чем 70 графическими функциями. Графическая библиотека поддерживает многочисленные стили линий и заполнителей, а также различные текстовые шрифты, которые можно изменять по размерам, способу выравнивания, а также ориентировать их либо по горизонтали, либо по вертикали.

Эти функции находятся в библиотечном файле `graphics.lib`, а их прототипы – в файле заголовка `graphics.h`. Кроме этих двух файлов, в состав графического пакета входят драйверы графических устройств (файлы `*.BGI`) и символные шрифты (`*.CHR`-файлы).

Функции библиотеки `graphics`

Графические функции языка C/C++ делятся на несколько категорий:

- управления графической системой
- черчения и заполнения
- вывода текстов
- управления цветами
- обработки ошибок

– запроса состояния

Ниже в таблицах приводится краткое описание некоторых графических функций. Более подробное описание этих функций приведено в документации по языку C/C++.

Таблица 1.1.

Функции управления графическим режимом.

Функция	Назначение функции
<code>closegraph()</code>	Закрывает графический режим
<code>detectgraph()</code>	Проверяет аппаратное обеспечение и определяет, какие графические драйверы использовать, рекомендует предпочтительный режим
<code>getgraphmode()</code>	Возвращает текущий графический режим
<code>initgraph()</code>	Инициализирует графическую систему и переводит аппаратное обеспечение в графический режим
<code>setgraphmode()</code>	Выбирает заданный графический режим, очищает экран и восстанавливает все умолчания

Для запуска графической системы необходимо прежде всего вызвать функцию `initgraph()`. Эта функция загружает графический драйвер и переводит систему в графический режим. Можно указать `initgraph()` использование конкретного графического драйвера и конкретный режим, либо задать автодетектирование установленного видео адаптера и выбор соответственного драйвера уже во время выполнения. Если задано в `initgraph()` автодетектирование, то она сама вызовет функцию `detectgraph()` для выбора графического драйвера и режима.

Для завершения работы в графике, необходимо вызвать функцию `closegraph()` для того, чтобы закрыть графическую режим.

Таблица 1.2.

Функции черчения

Функция	Назначение функции
arc()	Чертит дугу окружности
circle()	Чертит окружность
drawpoly()	Чертит контур многоугольника
ellipse()	Чертит эллиптическую дугу
line()	Чертит линию из (x0,y0) в (x1,y1)
lineto()	Чертит линию из текущей позиции в (x,y)
moveto()	Перемещает текущую позицию в (x,y)
rectangle()	Рисует прямоугольник
setlinestyle()	Устанавливает ширину и стиль текущей линии

Таблица 1.3.

Функции заполнения

Функция	Назначение функции
bar()	Чертит и заполняет столбик
bar3d()	Чертит и заполняет трехмерный столбик
fillellipse()	Чертит и заполняет эллипс
fillpoly()	Чертит и заполняет многоугольник
pieslice()	Чертит и заполняет сектор окружности
sector()	Чертит и заполняет эллиптический сектор
setfillpattern()	Выбирает шаблон заполнения, определяемый пользователем
setfillstyle()	Устанавливает шаблон и цвет заполнения

Таблица 1.4.

Функции получения информации о цвете

Функция	Назначение функции
getbcolor()	Возвращает текущий цвет фона
getcolor()	Возвращает текущий цвет вычерчивания
getmaxcolor()	Возвращает максимальное значение цвета, доступное в текущем графическом режиме

Таблица 1.5.

Функции установки одного или более цветов

Функция	Назначение функции
setbkcolor()	Устанавливает текущий цвет фона
setcolor()	Устанавливает текущий цвет вычерчивания

Таблица 1.6.

Обработка ошибок в графическом режиме

Функция	Назначение функции
grapherrormsg()	Возвращает строку с сообщением об ошибке для заданного кода ошибки
graphresult()	Возвращает код ошибки для последней графической операции, в которой встретилась ошибка

Таблица 1.7.

Функции запроса состояния

Функция	Назначение функции
getdrivername()	Возвращает имя текущего графического драйвера
getgraphmode()	Возвращает текущий графический режим
getlinesettings()	Возвращает текущие стиль, шаблон и толщину линии
getmaxx()	Возвращает текущее разрешение по оси x
getmaxy()	Возвращает текущее разрешение по оси y
getpixel()	Возвращает цвет пикселя в (x,y)
getx()	Возвращает координату x текущей позиции (CP)
gety()	Возвращает координату y текущей позиции (CP)

Пример.

Окружность радиусом 100 пикселей перемещается от левого края экрана к правому с шагом 10 пикселей, изменяя при этом цвет случайным образом.

```

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>
void main()
{
    /* описание переменных для графического режима с
автоопределением параметров */
    int graphdriver = ДЕТЕСТ, gmode, errorcode;
    int x, maxx, midy, color;
    int radius = 100;
    /* инициализация графики и локальных переменных */
    initgraph(&graphdriver, &gmode, "");
    /* получение результата инициализации */
    errorcode = graphresult();
    if(errorcode != grOk) /* если ошибка */
    {
        printf("Ошибка:                %s\n",
grapherrormessage(errorcode));
        printf("Для останова нажмите любую клавишу\n");
        getch();
        exit(1); /* завершение с кодом ошибки */
    }
    midy = getmaxy() / 2; /* определение середины
экрана по вертикали */
    maxx = getmaxx(); /* определение количества
пикселей по горизонтали */
    for (x= radius; x <= maxx - radius; x+=10)
    {
        color = random(getmaxcolor()); /* задание
случайного значения цвета */
        setcolor(color);
        /* рисование окружности */
        circle(x, midy, radius);
        delay(700); /* Задержка выполнения программы на
0,7 сек. */
    }
    closegraph(); /* закрытие графического режима */
}

```

Задание

Написать и отладить программу на языке C/C++ для вывода по заданным координатам и радиусу(ов) на экран графической фигуры. Для этого:

1. Изучить необходимые стандартные функции языка C/C++ для построения необходимых математических фигур. Построение многоугольников любой сложности реализовывать отрезками прямых линий.
2. Написать функцию для вывода на экран заданной по варианту графической фигуры (табл. А.1, приложение А). В функцию передавать минимально необходимый набор параметров (Н-р: координаты центра и один или два радиуса, в зависимости от вида фигуры).
3. Нарисовать заданную по варианту фигуру (табл. А.1, приложение А) в координатной плоскости по координатам центра и радиуса(ов). Предусмотреть ввод координат и радиуса(ов) вручную и случайным образом.

Содержание отчета

1. Титульный лист.
2. Задание.
3. Краткие теоретические сведения по графическим функциям языка C/C++.
4. Графическая фигура по варианту с указанием необходимых для построения параметров.
5. Примеры расчета координат вершин многоугольника и/или других точек необходимых для построения фигуры.
6. Текст программы.

При защите лабораторной работы тестирование программы **обязательно!**

Лабораторная работа № 2

Тема: координатные преобразования на плоскости

Простейшие двумерные преобразования

Точки на плоскости задаются с помощью двух ее координат. Таким образом, геометрически каждая точка задается значениями координат вектора относительно выбранной системы координат.

Точки на xu -плоскости можно перенести в новые позиции путем добавления к координатам этих точек констант переноса.

$$x' = x + Dx, \quad y' = y + Dy. \quad (2.1), (2.2)$$

Координаты точек можно рассматривать как элементы матрицы $[x, y]$, т. е. в виде вектор-строки или вектор-столбца. Определяя векторы-строки:

$$P = [xy], \quad P' = [x' y'], \quad T = [Dx Dy], \quad (2.3)-(2.5)$$

можно переписать это уравнение в векторной форме или более кратко:

$$[x' \ y'] = [x \ y] + [Dx \ Dy] = [x + Dx \ y + Dy]. \quad (2.6)$$

$$P' = P + T. \quad (2.7)$$

Т.е. положением этих точек можно управлять путем преобразования матрицы. Таким образом, для перемещения точки на плоскости надо к матрице ее координат прибавить матрицу коэффициентов преобразования. Все точки, принадлежащие отрезку, можно перенести путем перемещения одних лишь крайних точек отрезка и последующего вычерчивания нового отрезка между получившимися в результате точками. Это справедливо также для масштабирования (растяжения) и поворота.

Точки можно промасштабировать (растянуть) в Sx раз вдоль оси x ; и в Sy раз вдоль оси y , получив в результате новые точки, с помощью умножения

$$x' = x \cdot Sx, \quad y' = y \cdot Sy. \quad (2.8)-(2.9)$$

Определив матрицу S (2.10) можно записать (2.8), (2.9) в матричной форме (2.11):

$$S = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \quad (2.10)$$

$$[x' \ y'] = [x \ y] \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \quad (2.11)$$

или

$$P' = P \cdot S, \quad (2.12)$$

Отметим, что масштабирование производится относительно начала координат и примененное к началу координат, не приведет к изменению координат точки (0,0).

Точки могут быть повернуты на угол θ относительно начала координат. В общем виде математически поворот определяется следующим образом:

$$x' = x \cdot \cos \theta - y \cdot \sin \theta, \quad (2.13)$$

$$y' = x \cdot \sin \theta + y \cdot \cos \theta. \quad (2.14)$$

Матрица преобразования общего вида записывается так:

$$\begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \quad (2.15)$$

В матричной форме мы имеем

$$[x' \ y'] = [x \ y] \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \quad (2.16)$$

или

$$P' = P \cdot R, \quad (2.17)$$

где через R обозначена матрица поворота.

Для частных случаев. Поворот на 90° можно осуществить с помощью матрицы преобразования

$$R = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \quad (2.18)$$

Поворот на 180° получается с помощью матрицы

$$R = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \quad (2.19)$$

Отображение

В то время как чистое двумерное вращение в плоскости xu осуществляется вокруг оси, перпендикулярной к этой плоскости, отображение определяется поворотом на 180° вокруг оси, лежащей в плоскости xu .

Такое вращение вокруг линии $y = x$ происходит при использовании матрицы:

$$R = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad (2.20)$$

Однородные координаты и матричное представление двумерных преобразований

В однородных координатах точка $P(x, y)$ записывается как $P(W \cdot x, W \cdot y, W)$ для любого масштабного множителя $W \neq 0$. При этом если для точки задано ее представление в однородных координатах $P(X, Y, W)$, то можно найти ее двумерные декартовы координаты как $x = X/W$ и $y = Y/W$. Если W будет равно 1, операция деления не требуется. Однородные координаты можно представить как вложение промасштабированной с коэффициентом W двумерной плоскости в плоскость $z = W$ (здесь $z = 1$) в трехмерном пространстве.

Уравнения переноса записываются в виде матрицы преобразования в однородных координат следующим образом:

$$[x' \ y' \ 1] = [x \ y \ 1] \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Dx & Dy & 1 \end{bmatrix} \quad (2.21)$$

$$P' = P \cdot T(Dx, Dy), \quad (2.22)$$

$$T(Dx, Dy) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Dx & Dy & 1 \end{bmatrix}. \quad (2.23)$$

Уравнения масштабирования в матричной форме в однородных координат записываются в виде:

$$[x' \ y' \ 1] = [x \ y \ 1] \cdot \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (2.24)$$

Определяя

$$S(S_x, S_y) = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad (2.25)$$

имеем

$$P' = P \cdot S(S_x, S_y). \quad (2.26)$$

И, наконец, уравнения поворота можно представить в виде:

$$[x' \ y' \ 1] = [x \ y \ 1] \cdot \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (2.27)$$

Полагая

$$R(\theta) = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad (2.28)$$

имеем

$$P' = P \cdot R(\theta). \quad (2.29)$$

Основная матрица преобразования размером 3×3 для двумерных однородных координат может быть подразделена на четыре части:

$$\left[\begin{array}{cc|c} a & b & p \\ c & d & q \\ m & n & s \end{array} \right] \quad (2.30)$$

Как мы видим, a , b , c и d осуществляют изменение масштаба, сдвиг и вращение; m и n выполняют смещение, а p и q – получение проекций. Оставшаяся часть матрицы, элемент s , производит полное изменение масштаба. Чтобы показать это, рассмотрим преобразование

$$[X \ Y \ W] = [x \ y \ 1] \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \end{bmatrix} = [x \ y \ s]. \quad (2.31)$$

Здесь $X = x$, $Y = y$, а $W = s$. Это дает $x' = x/s$ и $y' = y/s$. В результате преобразования $[x \ y \ 1] \rightarrow [x/s \ y/s \ 1]$ имеет место однородное изменение масштаба вектора положения. При $s < 1$ происходит увеличение, а при $s > 1$ – уменьшение масштаба.

Композиция двумерных преобразований

Основное преимущество объединенных преобразований состоит в том, что к точке более эффективно применять одно результирующее преобразование, чем ряд преобразований друг за другом.

Рассмотрим, например, поворот объекта относительно некоторой произвольной точки P_i . Поскольку нам известно, как поворачивать вокруг начала координат, разобьем исходную проблему на три задачи:

1. Перенос, при котором точка P_i перемещается в начало координат.
2. Поворот.
3. Перенос, при котором точка из начала координат возвращается в первоначальное положение P_i .

Эта последовательность показана на рис. 2.1, на котором вокруг точки $P_i(x, y)$ поворачивается контур домика. Первый перенос производится на $(-x_i, -y_i)$, в то время как последующий – на (x_i, y_i) – является обратным ему. Результат существенно отличается от того, который получился бы, если бы применялся один только поворот.

Результирующее преобразование имеет вид

$$\begin{aligned} & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_i & -y_i & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_i & y_i & 1 \end{bmatrix} = \\ & = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ x_i(1-\cos \theta) + y_i \sin \theta & y_i(1-\cos \theta) - x_i \sin \theta & 1 \end{bmatrix}. \end{aligned} \quad (2.32)$$

Эта композиция преобразований путем умножения матриц служит примером того, как применение однородных координат упрощает задачу.

Используя аналогичный подход, можно промасштабировать объект относительно произвольной точки P_i : перенести P_i в начало координат,

промасштабировать, перенести назад в точку P_i . Результирующее преобразование в этом случае будет иметь вид

$$\begin{aligned} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_1 & -y_1 & 1 \end{bmatrix} \cdot \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_1 & y_1 & 1 \end{bmatrix} = \\ = \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ x_1(1-Sx) & y_1(1-Sy) & 1 \end{bmatrix}. \end{aligned} \quad (2.33)$$

Предположим, что нам необходимо промасштабировать, повернуть и расположить в нужном месте домик, показанный на рис. 2.1, где центром поворота и масштабирования является точка P_1 . В структуре данных, в которой содержится это преобразование, могут находиться масштабный множитель (множители), угол поворота и величины переноса или может быть записана матрица результирующего преобразования:

$$T(-x_1, -y_1) \cdot S(Sx, Sy) \cdot R(\theta) \cdot T(x_1, y_1). \quad (2.34)$$

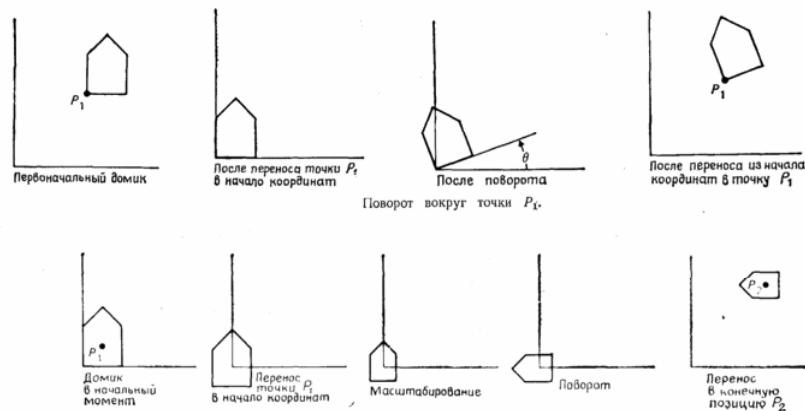


Рисунок 2.1. – Композиция преобразований

Для отражения объекта относительно произвольной прямой необходимо выполнить следующие действия:

- перемещение линии и объекта, таким образом, чтобы линия прошла через начало координат;
- поворот линии и объекта вокруг точки начала координат до совпадения с одной из координатных осей;
- отражение относительно координатной оси;
- обратный поворот вокруг начала координат;
- перемещение в исходное положение.

Задание

Написать и отладить программу на языке C/C++ для реализации всех видов координатных преобразований графической фигуры. Для этого:

1. Написать функцию для вывода на экран заданной по варианту графической фигуры (табл. А.1, приложение А). В функцию передавать минимально необходимый набор параметров. Построение многоугольников реализовывать отрезками прямых линий.
2. Нарисовать заданную по варианту фигуру в координатной плоскости по координатам вершин и / или радиуса и центра. Предусмотреть ввод координат вручную и случайным образом.
3. Написать функции для реализации заданных по варианту преобразований (табл. 2.1). Предусмотреть ввод с клавиатуры констант масштабирования по обеим осям, констант переноса по обеим осям, угол поворота.

Таблица 2.1. Варианты заданий.

№ варианта	Перемещение относительно оси:	Масштабирование относительно:	Поворот относительно:	Отображение относительно:
1	X	Центра фигуры	Начала координат	оси X
2	Y	Начала координат	Центра фигуры	оси Y
3	X	Центра фигуры	Начала координат	линии $y = const$
4	X	Начала координат	Центра фигуры	линии $y = x$
5	Y	Центра фигуры	Начала координат	линии $x = const$

6	X	Начала координат	Центра фигуры	Начала координат
7	X	Центра фигуры	Начала координат	оси Y
8	Y	Начала координат	Центра фигуры	линии $y = const$
9	X	Начала координат	Начала координат	линии $y = x$
10	Y	Центра фигуры	Центра фигуры	линии $y = x$
11	X	Центра фигуры	Начала координат	линии $y = const$
12	X	Начала координат	Центра фигуры	Начала координат
13	Y	Центра фигуры	Начала координат	линии $x = const$
14	Y	Начала координат	Центра фигуры	Начала координат
15	X	Начала координат	Начала координат	линии $y = const$
16	Y	Центра фигуры	Центра фигуры	линии $x = const$
17	X	Центра фигуры	Начала координат	Начала координат
18	Y	Начала координат	Центра фигуры	линии $y = x$
19	X	Центра фигуры	Начала координат	линии $y = const$

20	X	Начала координат	Центра фигуры	линии $x = const$
21	Y	Начала координат	Начала координат	оси X
22	Y	Центра фигуры	Центра фигуры	линии $x = const$
23	X	Центра фигуры	Начала координат	оси Y
24	Y	Начала координат	Центра фигуры	линии $y = const$

Все преобразования выполнять в однородных координатах. Реализовать функции пересчета декартовых координат в однородные и однородных координат в декартовы.

Содержание отчета

1. Титульный лист.
2. Задание.
3. Краткие теоретические сведения по графическим функциям языка C/C++.
4. Краткие теоретические сведения по координатным преобразованиям.
5. Графическая фигура по варианту с указанием необходимых для построения параметров.
6. Примеры расчета координат вершин многоугольника и/или других точек необходимых для построения фигуры.
7. Текст программы.

При защите лабораторной работы тестирование программы **обязательно!**

Лабораторная работа № 3.

Тема: координатные преобразования в пространстве, проекции

Трехмерные матричные преобразования

Окружающий нас мир с точки зрения практических приложений описывают как трехмерное евклидово пространство. Под описанием трехмерного объекта будем понимать знание о положении каждой точки объекта в пространстве в любой момент времени. Положение точек в пространстве удобно описывается с помощью декартовой системы координат.

В общем случае оси системы координат могут располагаться под произвольными, хотя и фиксированными углами друг относительно друга. Для практических расчетов гораздо удобнее, когда эти оси расположены взаимно перпендикулярно. Такая система координат называется ортогональной. Таким образом, положение в пространстве точки P описывается ее координатами, что записывается как $P = (x, y, z)$. Взаимное расположение осей в ортогональной системе координат в трехмерном пространстве может быть двух видов рис. 3.1.

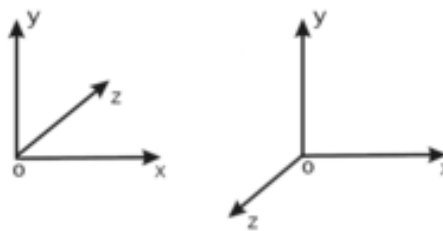


Рисунок 3.1. – Левосторонняя и правосторонняя системы координат.

В первом случае система координат будет называться левой или левосторонней, а во втором случае – правой или правосторонней.

Подобно тому, как двумерные преобразования описываются матрицами размером 3×3 , трехмерные преобразования могут быть представлены матрицами размером 4×4 . Тогда трехмерная точка (x, y, z)

записывается в однородных координатах как (wx, wy, wz, w) , где $w \neq 0$. Для получения декартовых координат надо первые три однородные координаты разделить на w . Преобразование из однородных координат описывается соотношениями (3.1), (3.2).

$$[X \ Y \ Z \ W] = [x \ y \ z \ 1] \quad (3.1)$$

$$[x^*, y^*, z^*, 1] = \left[\frac{X}{W}, \frac{Y}{W}, \frac{Z}{W}, 1 \right], W \neq 0. \quad (3.2)$$

Обобщенная матрица преобразования 4×4 для трехмерных однородных координат имеет вид

$$T = \begin{bmatrix} a & b & c & p \\ d & e & f & q \\ h & i & j & r \\ l & m & n & s \end{bmatrix} \quad (3.3)$$

Эта матрица может быть представлена в виде четырех отдельных частей:

$$\begin{bmatrix} 3 \times 3 & 3 \times 1 \\ 1 \times 3 & 1 \times 1 \end{bmatrix} \quad (3.4)$$

Матрица 3×3 осуществляет линейное преобразование в виде изменения масштаба, сдвига и вращения. Матрица-строка 1×3 производит перенос, а матрица-столбец 3×1 – преобразование в перспективе. Последний скалярный элемент выполняет общее изменение масштаба. Полное преобразование, полученное путем воздействия на вектор положения матрицей 4×4 и нормализации преобразованного вектора, будем называть билинейным преобразованием. Оно обеспечивает выполнение комплекса операций частичного изменения масштаба, вращения, отображения, переноса, а также изменения масштаба изображения в целом.

Матрицы преобразований будем записывать в **правосторонней системе координат**. При этом положительный поворот определяется следующим образом. Если смотреть из положительной части оси вращения (например, оси z) в направлении начала координат, то поворот на 90° против часовой стрелки будет переводить одну положительную полуось в другую (ось x в y , в соответствии с правилом циклической перестановки).

Таблица 3.1. Правосторонняя система координат

Если ось вращения	Положительным будет направление поворота
x	от y к z
y	от z к x
z	от x к y

Запишем теперь матрицу трехмерного переноса. Аналогично двумерному случаю.

$$T(D_x, D_y, D_z) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ D_x & D_y & D_z & 1 \end{bmatrix}, \quad (3.5)$$

при этом

$$[x, y, z, 1] \cdot T(D_x, D_y, D_z) = [x + D_x, y + D_y, z + D_z, 1]. \quad (3.6)$$

Операция масштабирования:

$$s(S_x, S_y, S_z) = \begin{bmatrix} S & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.7)$$

$$[x, y, z, 1] \cdot s(S_x, S_y, S_z) = [S_x \cdot x, S_y \cdot y, S_z \cdot z, 1] \quad (3.8)$$

Общее изменение масштаба получается за счет 4-го диагонального элемента, т. е.

$$s(S_x, S_y, S_z) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & S \end{bmatrix} \quad (3.9)$$

$$[x, y, z, 1] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & S \end{bmatrix} = [x, y, z, S] = \left[\frac{x}{S}, \frac{y}{S}, \frac{z}{S}, 1 \right] \quad (3.10)$$

Такой же результат можно получить при равных коэффициентах частичных изменений масштабов. В этом случае матрица преобразования такова:

$$S(S_x, S_y, S_z) = \begin{bmatrix} \frac{1}{S} & 0 & 0 & 0 \\ 0 & \frac{1}{S} & 0 & 0 \\ 0 & 0 & \frac{1}{S} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.11)$$

Перейдем к операции поворота. Так как при двумерном повороте в плоскости xy координаты z остаются неизменными, то поворот вокруг оси z записывается так:

$$R_z(\alpha) = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (3.12)$$

Матрица поворота вокруг оси x имеет вид:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (3.13)$$

и вокруг оси y :

$$R_y(\alpha) = \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.14)$$

Отображение в пространстве. В трех измерениях наиболее просто отображение осуществляется относительно плоскости. Для отображения без изменения масштабов необходимо, чтобы определитель преобразования был равен $-1,0$. При отображении относительно плоскости xy изменяется только знак координаты z . Следовательно, матрица преобразования для отображения относительно плоскости xy имеет вид:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.15)$$

Для отображения относительно плоскости yz :

$$T = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.16)$$

а для отображения на плоскости xz :

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.17)$$

Проецирование

Проецирование – отображение точек, заданных в системе координат с размерностью N , в точки в системе меньшей размерности.

Проекторы (проецирующие лучи) – отрезки прямых, идущие из центра проекции через каждую точку объекта до пересечения с плоскостью проекции.

Проекцией точки на координатную ось называется точка пересечения плоскости, проходящей через заданную точку и параллельной плоскости, образованной двумя другими осями координат. Например, на рис. 3.2 проекцией точки P на ось Ox является точка Q , которая принадлежит плоскости, параллельной плоскости zOy .

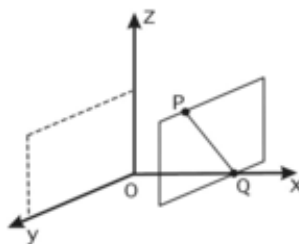


Рисунок 3.2. – Нахождение координаты $x=Q$ точки P .

Тип проецирования на плоскую, а не искривленную поверхность, где в качестве проекторов используются прямые, а не искривленные линии, называется плоской геометрической проекцией. На рис. 3.3 и 3.5 приведена классификация плоских геометрических проекций. Итак, они делятся на два вида: центральные (перспективные) и параллельные.

- При параллельном проектировании считается, что центр лучей (прямых) бесконечно удален, а прямые параллельны.
- При центральном проектировании (перспективном) все прямые исходят из одной точки, центр проекции находится на конечном расстоянии от проекционной плоскости.

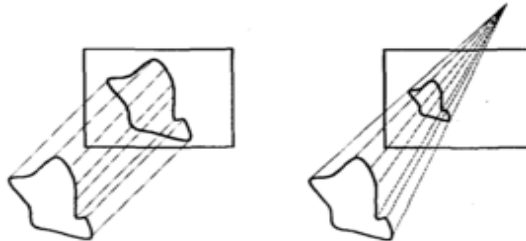


Рисунок 3.3. – Основные типы проекций

Центральная проекция приводит к визуальному эффекту, подобному тому, который дает зрительная система человека. При этом наблюдается эффект перспективного укорачивания, когда размер проекции объекта изменяется обратно пропорционально расстоянию от центра проекции до объекта.

Точкой схода называется точка пересечения центральных проекций любой совокупности параллельных прямых, которые не параллельны проекционной плоскости. Существует бесконечное множество точек схода. Точка схода называется главной если совокупность прямых параллельна одной из координатных осей. В зависимости от того, сколько координатных осей пересекает проекционную плоскость различают одно-, двух- и трехточечные проекции (рис.3.4).

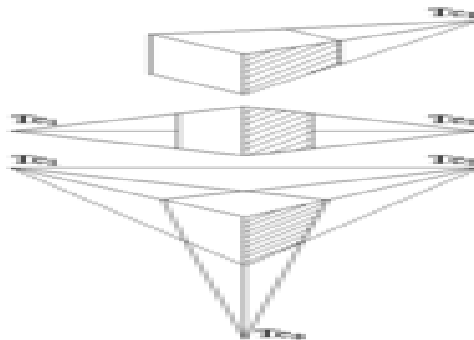


Рисунок 3.4. – Центральная проекция



Рисунок 3.5. – Классификация плоских проекций

В зависимости от соотношения между направлениями проецирования и нормалью к проекционной плоскости параллельные проекции разделяются:

- ортографические или ортогональные, в которых направление проецирования является нормалью к проекционной плоскости;
- косоугольные, в которых направление проецирования и нормаль к проекционной плоскости не совпадают.

В зависимости от положения осей системы координат объекта относительно проекционной плоскости ортографические проекции могут быть:

- параллельная прямоугольная проекция;
- аксонометрические проекции.

Простейшей проекцией является параллельная прямоугольная проекция. В ней совместно изображаются виды сверху, спереди и сбоку. Эти проекции часто используются в черчении.

При аксонометрической проекции проектирующие прямые перпендикулярны плоскости картинке. Различают изометрию, диметрию и триметрию.

Изометрия – все три угла между нормалью картинке и координатными осями равны.

Диметрия – два угла между нормалью картинке и координатными осями равны.

Триметрия – нормальный вектор плоскости картинке образует с координатными осями различные углы.

Каждый из этих трех видов этих проекций получается комбинацией поворотов, за которой следует параллельное проектирование.

Для косоугольных параллельных проекций лучи проецирования не перпендикулярны плоскости проецирования.

Математическое описание плоских геометрических проекций

Каждую из проекций можно описать матрицей 4×4 . Этот способ оказывается удобным, поскольку появляется возможность объединить матрицу проецирования с матрицей преобразования.

Центральная (перспективная) проекция получается путем перспективного преобразования и проецирования на некоторую двумерную плоскость «наблюдения». Перспективная проекция на плоскость $Z = 0$ обеспечивается преобразованием

$$[X \ Y \ Z \ H] = [x \ y \ z \ 1]^* \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & r \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x \ y \ 0 \ (rz+1)]. \quad (3.18)$$

или

$$x' = \frac{X}{H} = \frac{x}{rz+1}; \quad (3.19)$$

$$y' = \frac{Y}{H} = \frac{y}{rz+1}; \quad (3.20)$$

$$z' = \frac{Z}{H} = \frac{0}{rz+1}, \quad (3.21)$$

$$r = \frac{1}{k}. \quad (3.22)$$

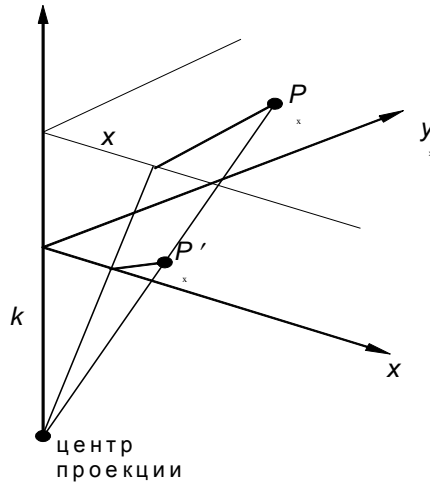


Рисунок 3.6. – Вычисление односточечной перспективы

Центр проекции находится в точке с координатами $(0,0,-k)$ (рис.3.6), плоскость проецирования $Z = 0$. Соотношения между x , y и x' , y' остаются теми же самыми. Рассматривая подобные треугольники, получим, что

$$\frac{x'}{k} = \frac{x}{z+k}, \quad (3.23)$$

или

$$x' = \frac{x}{\frac{z}{k}+1}; \quad (3.24)$$

аналогично

$$y' = \frac{y}{\frac{z}{k}+1}. \quad (3.25)$$

Координаты x' , y' являются преобразованными координатами. В перспективном проектировании преобразованное пространство не является евклидовым, так как ортогональность осей не сохраняется. При $k = \infty$ получим аксонометрическое преобразование.

Перспективному преобразованию может предшествовать произвольная последовательность аффинных преобразований.

Таким образом, чтобы получить перспективные изображения из произвольной точки наблюдения вначале используют аффинные преобразования, позволяющие сформировать систему координат с осью Z вдоль желаемой линии визирования. Затем применяется перспективное преобразование.

Выше было рассмотрено проецирование на плоскость $z = 0$, при этом теряется информация по координате z , которая может быть полезной при проверке неявно выраженных точек на плоскости. Для того чтобы не потерять информацию по координате z , можно использовать перспективное преобразование вида

$$[X \ Y \ Z \ H] = [x \ y \ z \ I]^* \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & r \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x \ y \ z \ (rz+I)]. \quad (3.26)$$

с координатами (3.19) – (3.21).

Мы рассмотрели односточечную перспективу с точкой схода Z ; если матрица преобразований имеет вид:

$$\begin{bmatrix} 1 & 0 & 0 & p \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (3.27)$$

то точка схода X

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & q \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (3.28)$$

Двухточечная (угловая) перспектива. Для получения двухточечной перспективы в общей матрице преобразования устанавливают коэффициенты p и q :

$$(x', y', z', I) = (x, y, z, I) \cdot \begin{bmatrix} 1 & 0 & 0 & p \\ 0 & 1 & 0 & q \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x, y, 0, (px+qy+I)]; \quad (3.29)$$

$$(x', y', z', I) = \left(\frac{x}{px+qy+1}, \frac{y}{px+qy+1}, 0, 1 \right). \quad (3.30)$$

Такое преобразование приводит к двум точкам схода. Одна расположена на оси X в точке $(\frac{1}{p}, 0, 0, 1)$, другая на оси Y в точке $(0, \frac{1}{q}, 0, 1)$.

Задание

Написать и отладить программу на языке C/C++ для реализации координатных преобразований в пространстве и получения проекций.

Постановка задачи:

1. Реализовать функцию для преобразования мировых координат в экранные.
2. Реализовать функцию для построения заданного тела (табл. Б.1, приложение Б). Построение многоугольников любой сложности реализовывать отрезками прямых линий.
3. Нарисовать заданное по варианту тело в пространстве.
4. Реализовать функции, выполняющие заданные по варианту (табл. 3.2) преобразования (перемещение вдоль оси, масштабирование, отражение относительно основных плоскостей, поворот на заданные с клавиатуры углы относительно указанных осей).
5. Реализовать функции, выполняющие переключение в различные виды проекций: ортогональные (на основные плоскости) и перспективные (одно-, двух- или трехточечные).

Таблица 3.2. Варианты заданий.

№ вар.	Перемещение и масштабирование	Отражение	Поворот	Ортогональная проекция	Перспективная проекция
1	X	XOY	Y	XOY	1
2	Z	XOZ	Y	XOY	2
3	Y	XOY	Y	XOZ	3
4	X	YOZ	X	XOZ	1
5	Y	XOY	X	XOZ	2

№ вар.	Перемещение и масштабирование	Отражение	Поворот	Ортогональная проекция	Перспективная проекция
6	Z	YOZ	X	XOZ	1
7	Y	YOZ	X	XOZ	3
8	X	XOZ	Z	YOZ	2
9	Z	XOY	Z	XOY	1
10	Y	XOY	Z	XOY	1
11	X	XOZ	Z	XOY	2
12	Z	XOY	X	XOZ	3
13	Y	YOZ	Y	XOY	1
14	X	XOY	Y	XOZ	2
15	Z	YOZ	X	XOY	1
16	Y	YOZ	X	XOZ	3
17	X	XOZ	Z	YOZ	2
18	Z	XOY	Y	XOY	1
19	Z	XOY	Y	XOY	1
20	Y	XOZ	X	XOY	2
21	X	XOY	Z	XOZ	3
22	X	YOZ	X	XOZ	1

Все преобразования выполнять в однородных координатах. Тело выводить без учета видимости / невидимости ребер (прорисовывать все ребра). Не выполнять заливку граней.

Содержание отчета

1. Титульный лист.
2. Задание.
3. Краткие теоретические сведения по координатным преобразованиям в пространстве и проекциям.
4. Пространственное тело по варианту с указанием необходимых для построения параметров.
5. Примеры расчета координат вершин и/или других точек необходимых для построения тела.
6. Текст программы.

При защите лабораторной работы тестирование программы **обязательно!**

Лабораторная работа № 4.

Тема: растровые алгоритмы построения контура фигур.

Алгоритм Брезенхема рисования линии

Пусть необходимо построить отрезок начало которого имеет координаты (x_1, y_1) , а конец (x_2, y_2) . Обозначим $dx = (x_2 - x_1)$, $dy = (y_2 - y_1)$. Не нарушая общности, будем считать, что начало отрезка совпадает с началом координат, и прямая имеет вид $y = \frac{dy}{dx}x$, где $\frac{dy}{dx} \in [0, 1]$, т.е. $dy < dx$. Считаем что начальная точка находится слева. Пусть на $(i-1)$ -м шаге текущей точкой отрезка является $P_{i-1} = (r, q)$. Необходимо сделать выбор следующей точки S_i или T_i .

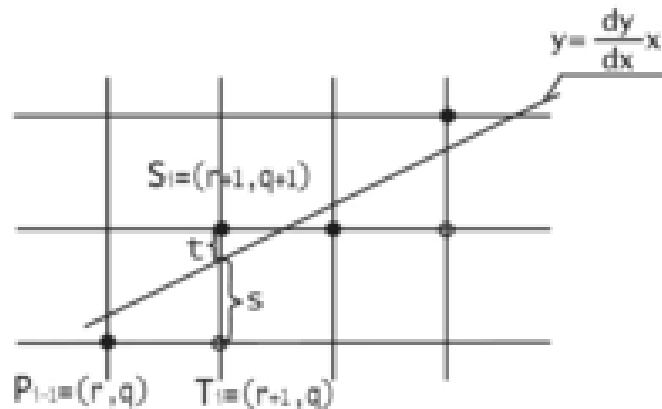


Рисунок 4.1. Рисование отрезков прямых по методу Брезенхема.

Алгоритм:

1. Берем первую точку $x_1 = x_{нач}$.
2. Рассчитываем $\Rightarrow d_1 = 2dy - dx$.
3. Если на предыдущем шаге $d_i < 0$, тогда $d_{i+1} = d_i + 2dy$ и $y_{i+1} = y_i + 1$. Если же на предыдущем шаге $d_i \geq 0$, то $d_{i+1} = d_i + 2(dy - dx)$ и $y_{i+1} = y_i$.
4. $x_i = x_i + 1$;
5. Если $x_i > x_{кон}$, то перейти на пункт 3, иначе – конец.

Блок схема алгоритма

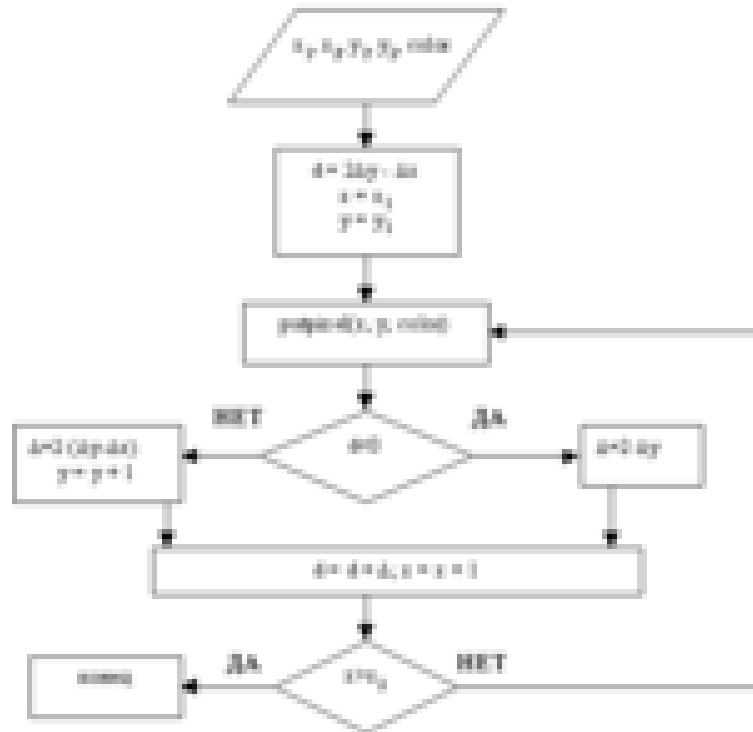


Рисунок 4.2 – Блок схема алгоритма Брезенхема для рисования линии
(для случая когда $dy < dx$).

Растровая развёртка окружности

Существует несколько очень простых, но не эффективных способов преобразования окружностей в растровую форму. Например, рассмотрим для простоты окружность с центром в начале координат. Ее уравнение записывается как

$$x^2 + y^2 = R^2. \quad (4.1)$$

Решая это уравнение относительно y , получим

$$y = \pm \sqrt{R^2 - x^2}. \quad (4.2)$$

Вторым простым методом растровой развертки окружности является использование параметрического представления окружности и вычислений x и y по формулам (4.3), (4.4)

$$x = R \cos \alpha, \quad (4.3)$$

$$y = R \sin \alpha. \quad (4.4)$$

при пошаговом изменении угла α от 0° до 360° .

Для упрощения алгоритма растровой развёртки стандартной окружности можно воспользоваться её симметрией относительно координатных осей и прямых $y = \pm x$; в случае, когда центр окружности не совпадает с началом координат, эти прямые необходимо сдвинуть параллельно так, чтобы они прошли через центр окружности. Тем самым достаточно построить растровое представление для $1/8$ части окружности, а все оставшиеся точки получить симметрией (рис. 4.3).

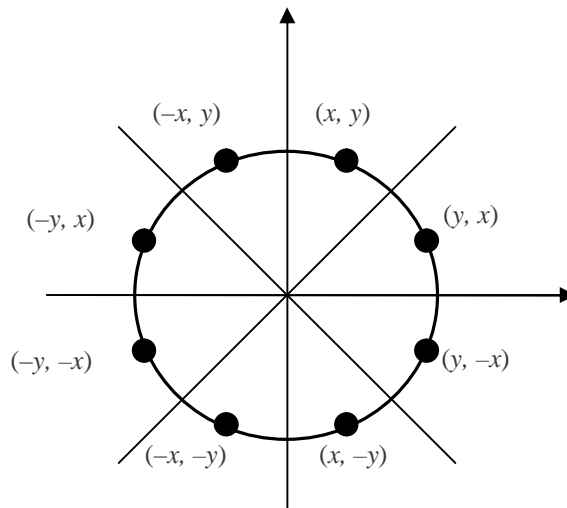


Рисунок 4.3. – Восьмисторонняя симметрия

Итак если точка (x, y) лежит на окружности, то легко вычислить семь точек, принадлежащих окружности, симметричных этой. То есть, имея функцию вычисления значения y по $x=0\dots R/\text{SQRT}(2)$ для построения дуги от 0° до 45° . Пример функции для вывода по координатам одной точки сразу восьми точек, симметричных центру окружности:

```
void Circle_Pixel(int x0, int y0, int x, int y, int
color);
{ putpixel(x0 + x, y0 + y, color);
  putpixel(x0 + y, y0 + x, color);
  putpixel(x0 + y, y0 - x, color);
  putpixel(x0 + x, y0 - y, color);
  putpixel(x0 - x, y0 - y, color);
  putpixel(x0 - y, y0 - x, color);
  putpixel(x0 - y, y0 + x, color);
  putpixel(x0 - x, y0 + y, color); }
```

Алгоритм Брезенхейма для окружности

Рассмотрим участок окружности из второго октанта $x \in [0, R/\sqrt{2}]$ (рис. 4.3). Опишем алгоритм Брезенхейма для этого участка окружности.

На каждом шаге алгоритм выбирает точку $P_i(x_i, y_i)$, которая является ближайшей к истинной окружности. Идея алгоритма заключается в выборе ближайшей точки при помощи управляющих переменных, значения которых можно вычислить в пошаговом режиме с использованием небольшого числа сложений, вычитаний и сдвигов.

Рассмотрим небольшой участок сетки пикселей, а также возможные способы (от А до Е) прохождения истинной окружности через сетку (рис. 4.4).

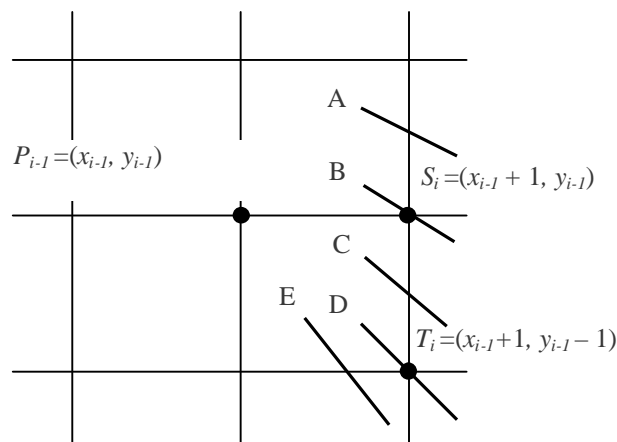


Рисунок 4.4. – Варианты прохождения окружности через растровую сетку

Предположим, что точка P_{i-1} была выбрана как ближайшая к окружности при $x = x_{i-1}$. Далее необходимо найти, какая из точек (S_i или T_i) расположена ближе к окружности при $x = x_{i-1} + 1$.

Алгоритм:

1. Берем первую точку $x_1 = x_{нач}$.
2. Рассчитываем $d_1 = 3 - 2R$.
3. Если $d_i < 0$, то $d_{i+1} = d_i + 4x_{i-1} + 6$ и выбирается S_i . Если $d_i \geq 0$, то $d_{i+1} = d_i + 4(x_{i-1} - y_{i-1}) + 10$ и выбирается T_i .
4. $x_i = x_i + 1$;
5. Если $x_i > x_{кон}$, то перейти на пункт 3, иначе – конец.

Блок-схема этого алгоритма

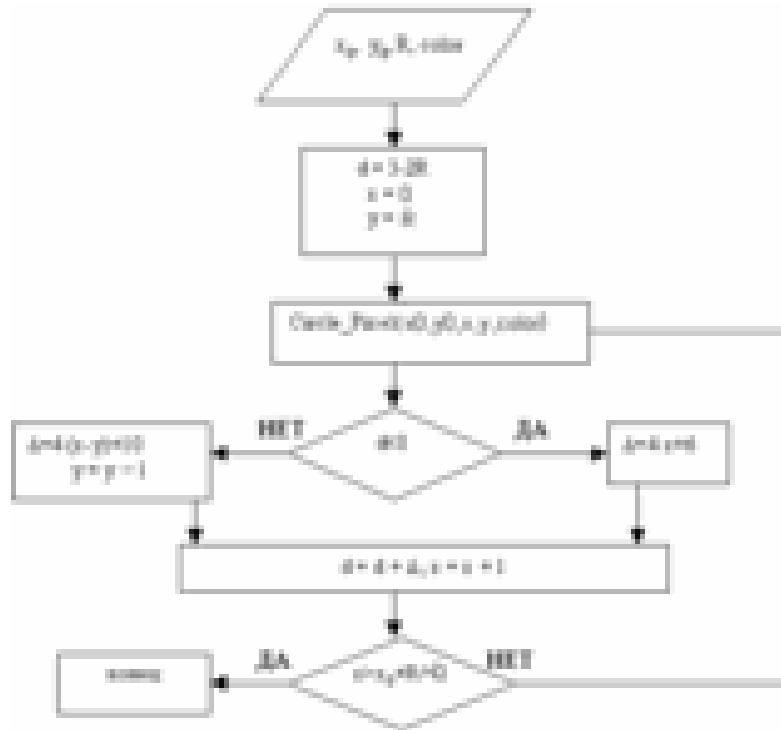


Рисунок 4.5. – Блок-схема алгоритма Брезенхейма для окружности.

Алгоритм хорош тем что отсутствуют операции с плавающей точкой, а также операции деления и извлечения корня.

Растровая развёртка эллипса

Простым методом растровой развертки эллипса является использование вычислений x и y по формулам

$$x = Rx \cos \alpha, \quad (4.5)$$

$$y = Ry \sin \alpha \quad (4.6)$$

при пошаговом изменении угла α от 0° до 360° .

Можно также использовать симметрию эллипса.

Существует модификация алгоритма Брезенхейма для эллипса.

Кривая Безье

Разработана математиком Пьером Безье. Кривые и поверхности Безье были использованы в 60-х годах компанией «Рено» для компьютерного

проектирования формы кузовов автомобилей. В настоящее время они широко используются в компьютерной графике.

Кривые Безье описываются в параметрической форме:

$$x = P_x(t), \quad y = P_y(t). \quad (4.7)$$

Значение t выступает как параметр, которому соответствуют координаты отдельной точки линии. Параметрическая форма описания может быть удобнее для некоторых кривых, чем задание в виде функции $y = ?(x)$, поскольку функция $?(x)$ может быть намного сложнее, чем $P_x(t)$ и $P_y(t)$, кроме того, $?(x)$ может быть неоднозначной.

Многочлены Безье для P_x и P_y имеют такой вид:

$$P_x(t) = \sum_{i=0}^m C_m^i t^i (1-t)^{m-i} x_i, \quad P_y(t) = \sum_{i=0}^m C_m^i t^i (1-t)^{m-i} y_i. \quad (4.8)$$

где x_i и y_i — координаты точек-ориентиров P_i , а величины C_m^i — это известные из комбинаторики, так называемые сочетания (они также известны как коэффициенты бинома Ньютона):

$$C_m^i = \frac{m!}{i!(m-i)!}. \quad (4.9)$$

Значение m можно рассматривать и как степень полинома, и как значение, которое на единицу меньше количества точек-ориентиров.

Рассмотрим кривые Безье, классифицируя их по значениям m .

1. $m = 1$ (по двум точкам)

Кривая вырождается в отрезок прямой линии, которая определяется конечными точками P_0 и P_1 , как показано на рис. 4.6:

$$P(t) = (1-t)P_0 + tP_1 \quad (4.10)$$

2. $m = 2$ (по трем точкам, рис. 4.6):

$$P(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2. \quad (4.11)$$

3. $m = 3$ (по четырем точкам, кубическая, рис. 4.7). Используется довольно часто, в особенности в сплайновых кривых:

$$P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t) P_2 + t^3 P_3, \quad (4.12)$$

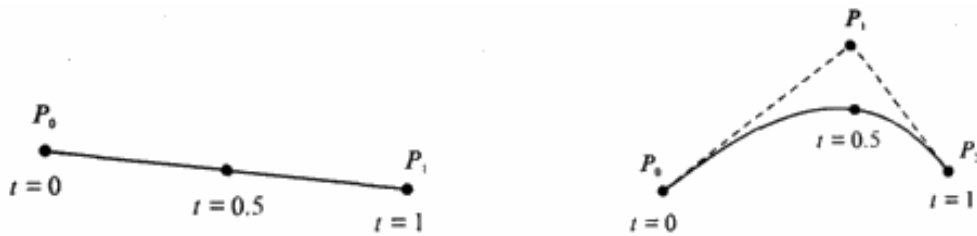


Рисунок 4.6. – Кривая Безье ($m=1$) и кривая Безье ($m=2$)

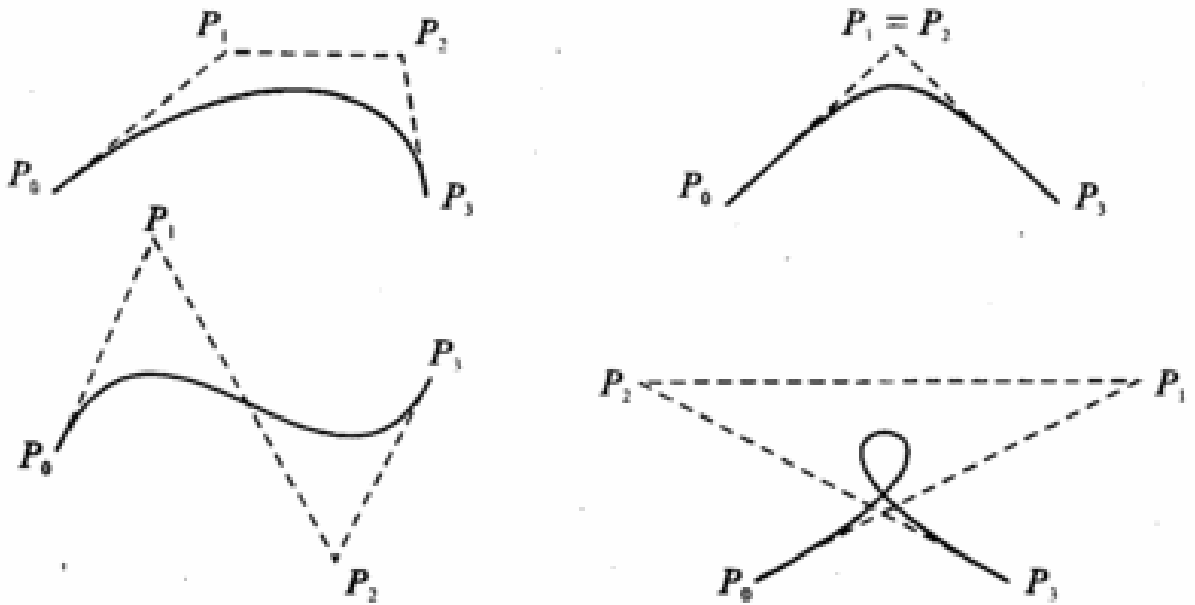


Рисунок 4.7. – Кубические кривые Безье ($m=3$)

Геометрический алгоритм для кривой Безье

Этот алгоритм позволяет вычислить координаты (x, y) точки кривой Безье по значению параметра t .

1. Каждая сторона контура многоугольника, который проходит по точкам-ориентирам, делится пропорционально значению t .

2. Точки деления соединяются отрезками прямых и образуют новый многоугольник. Количество узлов нового контура на единицу меньше, чем количество узлов предшествующего контура.

3. Стороны нового контура снова делятся пропорционально значению t . И так далее. Это продолжается до тех пор, пока не будет получена единственная точка деления. Эта точка и будет точкой кривой Безье (рис.4.8).

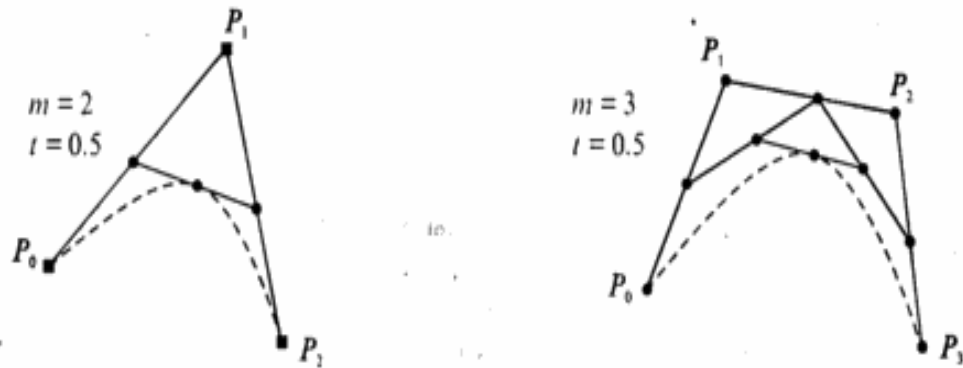


Рисунок 4.8. – Геометрический алгоритм для кривых Безье

Пример на языке C++:

```
int m=3; // степень полинома
int n=6; // количество точек на кривой
float t;
for (int l=0; l<=n; l++)
{
    t=(1.0/n)*l;
    for (int i=0; i<2; i++)
        for (int j=0; j<=m; j++)
            R[i][j]=(P[i][j]); // вспомогательный //
// массив дублирует координаты точек - ориентиров.
    for (int i=0; i<2; i++) // координаты x и y
        for (int j=m; j>=0; j--)
            for (int k=0; k<j; k++)
                { R[i][k]+= t*(R[i][k+1]-R[i][k]);
                  T[i][l]=R[i][k] ; //координаты точек на кривой
                }
}
```

Уравнения кривой Безье можно записать в матричном виде:

$$P(t) = [T][N][C] \quad (4.13)$$

где для $m = 3$:

$$P(t) = [T][N][C] = [t^3 \ t^2 \ t \ 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_3 \\ P_2 \\ P_1 \\ P_0 \end{bmatrix}. \quad (4.15)$$

Матрица N представляет собой коэффициенты при соответствующих t (Первый столбец при P_0 , второй – при P_1 , третий – при P_2 и четвертый – при P_3 . Первая строка при t^3 , вторая – при t^2 , третья – при t и четвертая – свободный член). P_0, P_1, P_2 и P_3 – вершины многоугольника Безье.

Аналогично для $m = 4$:

$$P(t) = [t_4 \ t_3 \ t_2 \ t \ 1] \begin{bmatrix} 1 & -4 & 6 & -4 & 1 \\ -4 & 12 & -12 & 4 & 0 \\ 6 & -12 & 6 & 0 & 0 \\ -4 & 4 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_3 \\ P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix} \quad (4.16)$$

В-сплайны

Кривая, представленная в виде кубического В-сплайна, в общем случае может проходить через любые управляющие точки, однако она непрерывна и, кроме того, непрерывностью изменения обладают ее касательный вектор и кривизна (т. е. первая и вторая производные кривой непрерывны в конечных точках) в отличие от формы Безье, у которой в конечных точках непрерывны лишь первые производные (но которые проходят через управляющие точки). Таким образом, можно утверждать, что форма В-сплайнов «более гладкая», чем другие формы. В-сплайн описывается следующей формулой:

$$x(t) = TM_i G_m \quad (4.17)$$

где

$$M_i = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \quad (4.18)$$

При аппроксимации управляющих точек P_1, P_2, \dots, P_n последовательностью В-сплайнов необходимо применять между каждой парой соседних точек геометрические матрицы. Для аппроксимации в интервале, близком к точкам P_i и P_{i+1} , используется

$$G_i = \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}, \quad 2 \leq i \leq n-2 \quad (4.19)$$

Пример 1. Построение дерева (рис. 4.9)

Точки – ориентиры для сегментов кривой Безье, сегменты нумеруются по часовой стрелке:

- 1) P= 4 0
 2 1
 2 3
 3.5 2.7
- 2) P= 3.5 2.7
 3 4.5
 3 7
 3.7 5
- 3) P= 3.7 5
 4 8
 4 8
 5 5
- 4) P= 5 5
 6 7
 6 5
 5 3
- 5) P= 5 3
 6 3.5
 6 1.5
 5 0

Пример 2. Построение цветка (рис.4.10).

Точки – ориентиры для сегментов кривой Безье:

- 1) P= 3 2
 2.5 2.5
 2.5 1.5
 3 2
- 2) P= 3 2
 3.5 2.5
 3.5 1.5
 3 2
- 3) P= 3 2
 2.5 2.5
 3.5 2.5
 3 2
- 4) P= 3 2
 2.5 1.5
 3.5 1.5
 3 2
- Стебель:
- 5) P= 3 0.75
 3.5 1.5
 2 1
 3 2

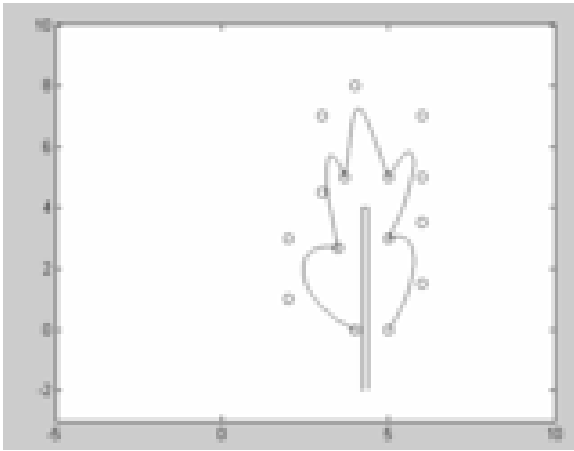


Рисунок 4.9. – Пример рисунка – дерево.

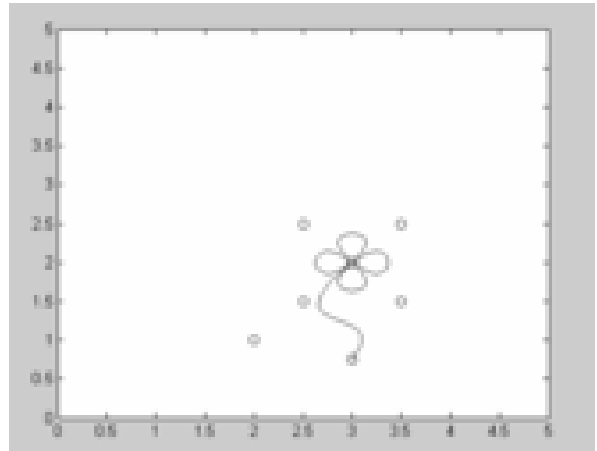


Рисунок 4.10. – Пример рисунка – цветок.

Задание

Написать и отладить программу на языке C/C++ для вывода рисунка по варианту (табл. В.1, приложение В). Для этого:

1. Написать необходимые функции для реализации растровой развертки графических примитивов:
 - отрезок (алгоритмом Брезенхема);
 - окружность (алгоритмом Брезенхема);
 - эллипс (параметрический);
 - сплайновые кривые (Безье и / или В-сплайны).
2. Нарисовать заданный по варианту рисунок (табл. В.1, приложение В).

Содержание отчета

1. Титульный лист.
2. Задание.
3. Описание методов построения примитивов. Для сплайновых кривых показать точки ориентиры и сегменты.
4. Текст программы.

При защите лабораторной работы тестирование программы **обязательно!**

Лабораторная работа № 5.

Тема: алгоритмы отсечения фигур.

Теоретические основы

Этап отсечения необходим и присутствует в большинстве задач компьютерной графики для определения реальных областей фигур, которые будут выведены на экран. Это необходимо, если фигура больше или выходит за пределы экрана или окна вывода. Отсечение чаще всего проводится для отбрасывания частей фигуры, выходящих за границу области, которая определяет экран или видимую область некоторого прямоугольного окна, хотя отсечение может проводиться и относительно произвольного многоугольника. Для отсечения могут использоваться как методы отсечения отрезка, так и методы отсечения многоугольников, которые, по сути, являются их модификацией. Если необходимо отсечь более сложную фигуру, то выполняется аппроксимация контура отрезками прямых. При отсечении фигуры порождается новый многоугольник или несколько новых многоугольников.

Рассмотрим эту задачу сначала применительно к отрезкам прямых. Некоторые из них полностью лежат внутри области экрана, другие целиком вне ее, а некоторые пересекают границу экрана. Правильное отображение отрезков означает нахождение точек пересечения их с границей экрана и рисование только тех их частей, которые попадают на экран.

Алгоритм Коэна-Сазерленда

Рассмотрим алгоритм Коэна-Сазерленда для отсечения отрезков прямых. Этот алгоритм позволяет легко определять нахождение отрезка полностью внутри или полностью снаружи окна, и если так, то его можно рисовать или не рисовать, не заботясь об отсечении по границе окна.

Для работы алгоритма вся плоскость в которой лежит окно разбивается на девять подобластей или квадратов, как показано на рис. 5.1.

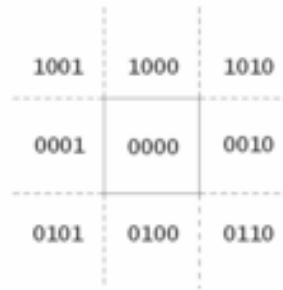


Рисунок 5.1. – Разбиение на подобласти в методе Коэна-Сазерленда.

Окну соответствует область обозначенная кодом 0000. Конечным точкам отрезка приписывается 4-битный код «вне/внутри» в зависимости от нахождения отрезка в соответствующей подобласти. Каждому биту присваивается значение 1 в соответствии со следующим правилом.

- Бит 1 – точка находится выше окна;
- Бит 2 – точка находится ниже окна;
- Бит 3 – точка находится справа от окна;
- Бит 4 – точка находится слева от окна;

Иначе биту присваивается нулевое значение. Значения этих битов для конечных точек отрезков легко определить по знакам соответствующих разностей: $(y_{\max} - y)$ – для 1-го бита, $(y - y_{\min})$ – для 2-го бита, $(x_{\max} - x)$ – для 3-го бита и $(x - x_{\min})$ – для 4-го бита.

Для каждого отрезка рассчитываются коды концов (K_1 , K_2) затем производится экспресс анализ:

- если $K_1 \wedge K_2 \neq 0$, тогда отрезок лежит вне поля вывода – отрезок отбрасывается;
- если $K_1 = K_2 = 0$, тогда отрезок полностью лежит внутри поля вывода – отсечение не нужно, отрезок полностью прорисовывается;
- если $K_1 \wedge K_2 = 0$, отрезок может частично лежать внутри поля вывода – необходимо отсечение по полю вывода.

В этом случае применяется последовательное деление отрезка, так что на каждом шаге конечная точка отрезка с ненулевым кодом вне/внутри заменяется на точку, лежащую на стороне окна или на прямой содержащей сторону.

Когда $K_1 \wedge K_2 = 0$ необходимо отсекать отрезок по границам поля вывода, отсечение происходит последовательно по всем сторонам рис.5.2.

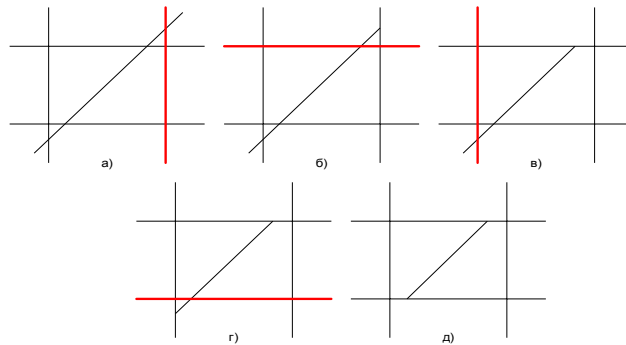


Рисунок 5.2. – Отсечение отрезка по прямоугольной области

На рис. 5.2 жирным выделено ребро по которому происходит отсечение. Также надо отметить, что точки лежащие на границе поля вывода принадлежат полю вывода.

На каждом шаге отсечения вычисляются новые координаты одной точки.

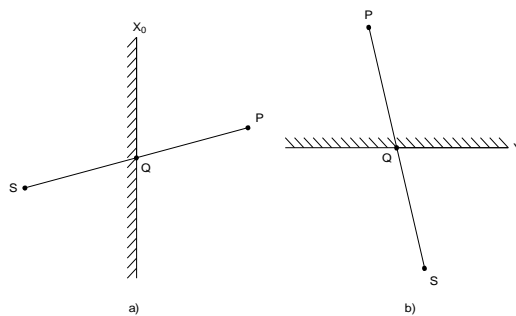


Рисунок 5.3. – Варианты пересечения отрезка и стороны окна.

Для нахождения координат точки $Q(x_Q, y_Q)$ на рис. 5.3 а) используется формула (5.1); для точки $Q(x_Q, y_Q)$ рис. 5.3 б) – формула (5.2).

$$x_Q = x_S + \frac{y_0 - y_S}{y_P - y_S} (x_P - x_S) \quad (5.1)$$

$$y_Q = y_S + \frac{x_0 - x_S}{x_P - x_S} (y_P - y_S) \quad (5.2)$$

Алгоритм Сазерленда-Ходгмана

Теперь рассмотрим операцию отсечения многоугольника. Решение общей задачи сводится к решению ряда простых и похожих подзадач. Примером такой подзадачи является отсечение многоугольника относительно одной отсекающей границы. Последовательное решение четырех таких задач позволяет провести отсечение относительно прямоугольной области (рис. 5.4).

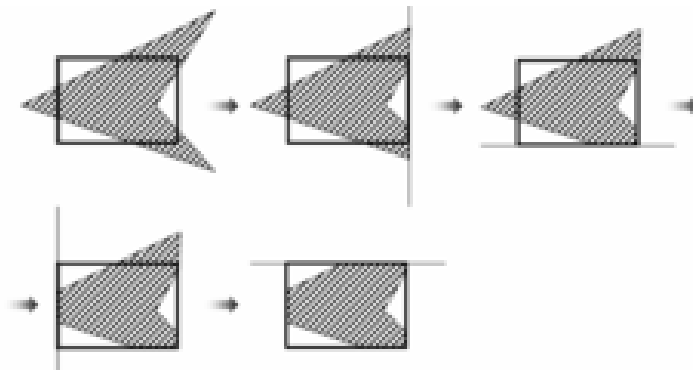


Рисунок 5.4. Последовательное отсечение многоугольника

На вход алгоритма поступает последовательность вершин многоугольника V_1, V_2, \dots, V_n . Ребра многоугольника проходят от V_i к V_{i+1} от V_n к V_1 . С помощью алгоритма производится отсечение относительно ребра и выводится другая последовательность вершин, описывающая усеченный многоугольник.

Алгоритм «обходит» вокруг многоугольника от V_n к V_1 и обратно к V_n , проверяя на каждом шаге соотношение между последовательными вершинами и отсекающей границей. Необходимо проанализировать четыре случая рис. 5.5.

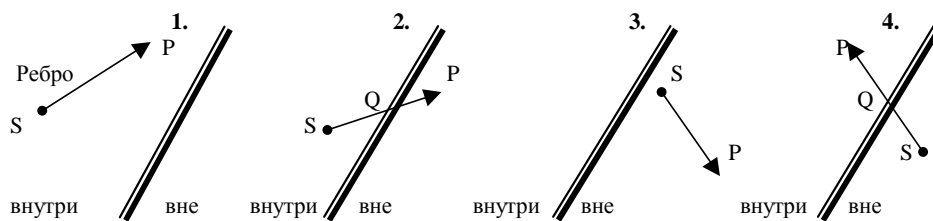


Рисунок 5.5. – Случаи, возникающие при отсечении многоугольников

Может также возникнуть вариант, когда ребро совпадает с границей.

Рассмотрим случаи на рис 5.5:

1. Добавить в список рисуемых вершин P;
2. Добавить в список Q.
3. Ничего не добавляется.
4. Добавить в список P,Q.

При отсечении многоугольника описанным алгоритмом возникает проблема, связанная с тем, что появляются ребра, частично совпадающие с границей окон. Лишние ребра можно устранить, введя дополнительную обработку или воспользовавшись более общим и более сложным алгоритмом Вейлера – Азерттона, который позволяет выполнять отсечение произвольного многоугольника по произвольному многоугольнику.

Задание

Написать и отладить программу на языке C/C++ для вывода объекта по полю вывода (табл. Д.1, приложение Д). Для этого:

1. Написать необходимые функции для отсечения многоугольника или отрезка по полю вывода.
2. Выполнить задания согласно варианту (табл. Д.1 в приложении Д):
 - нарисовать многоугольник или отрезок;
 - показать зону видимости;
 - перемещать многоугольник или отрезок по экрану, при прохождении через зону видимости выделять видимую часть другим цветом.

Содержание отчета

1. Титульный лист.
2. Задание.
3. Описание метода отсечения.
4. Текст программы.

При защите лабораторной работы тестирование программы **обязательно**

Лабораторная работа № 6.

Тема: растровые алгоритмы заполнения фигур.

Теоретические основы

Алгоритмы заполнения (закраски) областей делятся на группы: растровой развертки и затравочного заполнения.

В методах растровой развертки область представляет собой многоугольник с заданными координатами концов, ограничивающих его отрезков. Этими методами находится пересечение строк с многоугольником, определяются пары точек, ограничивающих закрашиваемый отрезок, и на основе этого осуществляется закрашка.

В методах затравочного заполнения область задается граничными пикселями, которые окрашены в определенный цвет. Кроме того, задается некоторый пиксель внутри области – затравочный. Затем ищутся пиксели, соседние с затравочным, и закрашиваются.

По способу задания области делятся на два типа:

- а) гранично-определенные, задаваемые своей (замкнутой) границей такой, что коды пикселей границы отличны от кодов внутренней, перекрашиваемой части области. На коды пиксели внутренней части области налагаются два условия – они должны быть отличны от кода пикселей границы и кода пикселя перекраски. Если внутри гранично-определенной области имеется еще одна граница, нарисованная пикселями с тем же кодом, что и внешняя граница, то соответствующая часть области не должна перекрашиваться;
- б) внутренне-определенные, нарисованные одним определенным кодом пикселя. При заполнении этот код заменяется на новый код закрашки.

Заполняемая область или ее граница – некоторое связное множество пикселей. По способам доступа к соседним пикселям области делятся на 4- и 8- связные. В 4-связных областях доступ к соседним пикселям

осуществляется в четырех направлениях - горизонтально влево и вправо и вертикально вверх и вниз. В 8-связных областях к этим направлениям добавляются еще 4 диагональных. Используя тот или иной тип связности, мы можем, двигаясь от точки затравки, закрасить все пиксели области.

Важно отметить, что для 4-связной прямоугольной области граница 8-связна и наоборот у 8-ми связной области граница 4-х связна. Поэтому заполнение 4-связной области 8-связным алгоритмом может привести к «просачиванию» через границу и закраске пикселей в примыкающей области.

В общем, 4-связную область мы можем заполнить как 4-, так и 8-связным алгоритмом. Обратное же неверно.

Закраска области, заданной цветом границы

Рассмотрим алгоритмы закрашивания произвольного контура, который уже нарисован в растре. Сначала определяются координаты произвольного пикселя, находящегося внутри очерченного контура фигуры. Цвет этого пикселя изменяем на нужный цвет заполнения. Потом проводится анализ цветов всех соседних пикселей. Если цвет некоторого соседнего пикселя не равен цвету границы контура или цвету заполнения, то цвет этого пикселя изменяется на цвет заполнения. Потом анализируется цвет пикселей, соседних с предшествующими. И так далее, пока внутри контура все пиксели не перекрасятся в цвет заполнения.

Пиксели контура образуют границу, за которую нельзя выходить в ходе последовательного перебора всех соседних пикселей. Соседними могут считаться только четыре пикселя (сосед справа, слева, сверху и снизу – четырехсвязность), или восемь пикселей (восьми-связность).

Простейший алгоритм закрашивания. Для всех алгоритмов закрашивания нужно задавать начальную точку внутри контура с координатами x_0 , y_0 . Простейший алгоритм можно описать всего двумя шагами.

Простейший рекурсивный алгоритм:

```
void PixelFill (int x, int y, int border_color, int
color)
{
    int c = getpixel(x,y);
    if ((c!=border_color)&&(c!=color))
    {
        putpixel(x, y , color);
        PixelFill (x-1, y , border_color, color);
        PixelFill (x+1, y , border_color, color);
        PixelFill (x, y-1 , border_color, color);
        PixelFill (x, y+1 , border_color, color);
    }
}
```

Этот алгоритм является слишком неэффективным, так как для всякого уже отрисованного пикселя функция вызывается ещё 4 раза и, кроме того, данный алгоритм не пригоден для закрашивания контуров фигур площадью в тысячу и более пикселей, так как вложенные вызовы функций делаются для каждого пикселя, что приводит к переполнению стека в ходе выполнения программы.

Поэтому для решения задачи закраски области предпочтительнее алгоритмы, способные обрабатывать сразу целые группы пикселей, т. е. использовать их «связность». Если данный пиксель принадлежит области, то, скорее всего, его ближайшие соседи также принадлежат данной области. Группой таких пикселей обычно выступает полоса, определяемая правым пикселем.

Растровое заполнение круга и полигона

Его основная идея – закрашивание фигуры отрезками прямых линий. Удобней использовать горизонталы. Алгоритм представляет собою цикл вдоль оси y , в ходе этого цикла выполняется поиск точек пересечения линии контура с соответствующими горизонталями.

Заполнение круга. Для заполнения круга можно использовать алгоритм вывода контура. В процессе выполнения этого алгоритма последовательно

вычисляются координаты пикселей контура в границах одного октанта. Для заполнения следует выводить горизонталы, которые соединяют пары точек на контуре, расположенные симметрично относительно оси y (рис. 6.2).

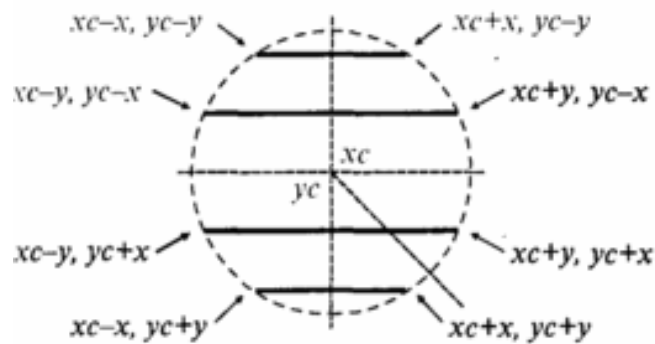


Рисунок 6.2. – Заполнение круга

```
void PixelFill_circle (int x0, int y0, int x, int y,
int color)
{
    int col=getcolor();
    setcolor(color);
    line(x0+x,y0-y,x0-x,y0-y);
    line(x0-y,y0-x,x0+y,y0-x);
    line(x0-y,y0+x,x0+y,y0+x);
    line(x0-x,y0+y,x0+x,y0+y);
    setcolor(col);
}
```

Так же можно создать и алгоритм заполнения эллипса.

Заполнение полигонов. Контур полигона определяется вершинами, которые соединены отрезками прямых – ребрами.

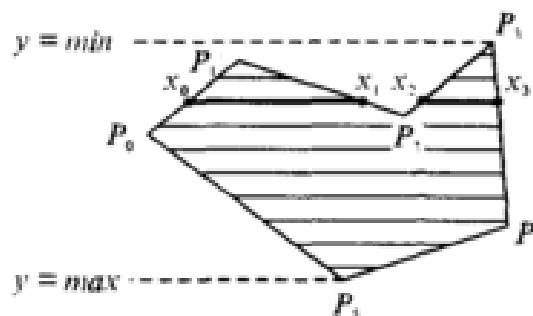


Рисунок 6.3. – Заполнение полигона

При нахождении точек пересечения горизонталы с контуром необходимо принимать во внимание особые точки. Если горизонталь имеет координату (y), совпадающую с координатой y_i вершины P_i тогда надлежит

анализировать то, как горизонталь проходит через вершину. Если горизонталь при этом пересекает контур, как, например, в вершинах P_0 или P_4 , то в массив записывается одна точка пересечения. Если горизонталь касается вершины контура (в этом случае вершина соответствует локальному минимуму или максимуму, как, например, в вершинах P_1 , P_2 , P_3 или P_5), тогда координата точки касания или не записывается, или записывается в массив два раза. Это является условием четного количества точек пересечения, хранящихся в массиве $\{x_j\}$.

Процедура определения точек пересечения контура с горизонталью, учитывая анализ на локальный максимум, может быть достаточно сложной.

Рассмотрим, какие случаи могут возникнуть при делении многоугольника на области сканирующей строкой.

1. Простой случай. Например, на рис. 6.4 сканирующая строка $y = 4$ пересекает многоугольник при $x = 1$ и $x = 6$. Получается три области: $x < 1$; $1 \leq x \leq 6$; $x > 6$. Сканирующая строка $y = 6$ пересекает многоугольник при $x = 1$; $x = 2$; $x = 5$; $x = 6$. Получается пять областей: $x < 1$; $1 \leq x \leq 2$; $2 < x < 5$; $5 \leq x \leq 6$; $x > 6$. В этом случае x сортируется в порядке возрастания. Далее список иксов рассматривается попарно. Между парами точек пересечения закрашиваются все пиксели. Для $y = 4$ закрашиваются пиксели в интервале $(1, 6)$, для $y = 6$ закрашиваются пиксели в интервалах $(1, 2)$ и $(5, 6)$.

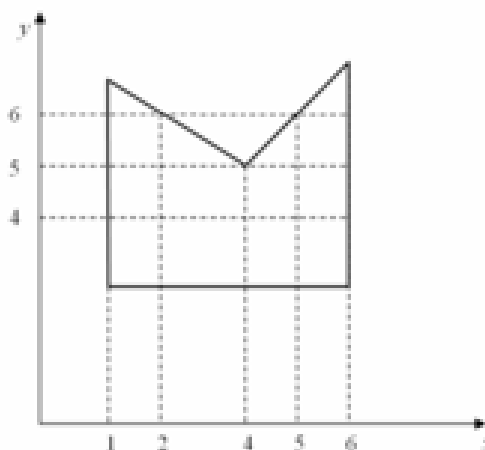


Рисунок 6.4. – Прохождение сканирующих строк по многоугольнику

2. Сканирующая строка проходит через вершину (рис. 6.5).

Например, по сканирующей строке $y = 3$ упорядоченный список x получится как $(2, 2, 4)$. Вершина многоугольника была учтена дважды, и поэтому закрашиваемый интервал получается неверным: $(2, 2)$. Следовательно, при пересечении вершины сканирующей строкой она должна учитываться единожды. И список по x в приведенном примере будет $(2, 4)$.

3. Сканирующая строка проходит через локальный минимум или максимум (рис. 6.4 при $y = 5$). В этом случае учитываются все пересечения вершин сканирующей строкой. На рис. 6.6 при $y = 5$ формируется список x $(1, 4, 4, 6)$. Закрашиваемые интервалы $(1, 4)$ и $(4, 6)$. Условие нахождения локального минимума или максимума определяется при рассмотрении концевых вершин для ребер, соединенных в вершине. Если y обоих концов координаты x больше, чем y вершины пересечения, то вершина – локальный минимум. Если меньше, то вершина пересечения – локальный максимум.

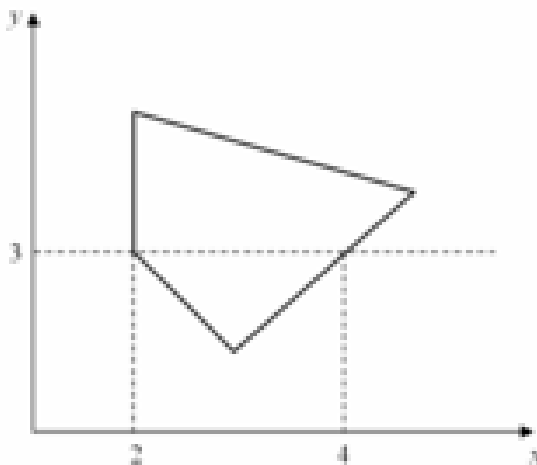


Рисунок 6.5. Прохождение сканирующей строки через вершину

Для ускорения работы алгоритма используется список активных ребер (САР). Этот список содержит те ребра многоугольника, которые пересекают сканирующую строку. При пересечении очередной сканирующей строки вершины многоугольника, из САР удаляются ребра, которые находятся выше, и добавляются концы, которые пересекает сканирующая строка. При работе алгоритма находятся пересечения сканирующей строки только с ребрами из САР.

Задание

Написать и отладить программу на языке C/C++ для вывода и заполнения объекта вывода (табл. Е.1, приложение Е). Для этого:

1. Написать необходимые функции для реализации растровых алгоритмов:
 - закрашка области, заданной цветом;
 - закрашка окружности (растровое заполнение);
 - закрашка эллипса (растровое заполнение);
 - закрашка многоугольника (растровое заполнение);
2. Выполнить задания по варианту согласно данным табл. Е.1 в приложении Е:
 - нарисовать и закрасить многоугольник;
 - нарисовать две окружности одна из них с полостью внутри;
 - закрасить окружности одну используя растровое заполнение, другую – затравочный метод;
 - нарисовать и закрасить эллипс;

Содержание отчета

1. Титульный лист.
2. Задание.
3. Описание реализованных методов.
4. Текст программы.

При защите лабораторной работы тестирование программы **обязательно!**

Лабораторная работа № 7.

Тема: фракталы.

Теоретические основы

Фракталы – это геометрические объекты с удивительными свойствами: любая часть фрактала содержит его уменьшенное изображение. То есть, сколько фрактал не увеличивай, из любой его части на вас будет смотреть его маленькая копия.

Первые идеи фрактальной геометрии возникли в 19 веке. Кантор с помощью простой рекурсивной (повторяющейся) процедуры превратил линию в набор несвязанных точек (так называемая Пыль Кантора). Он брал линию и удалял центральную треть и после этого повторял то же самое с оставшимися отрезками. Пеано нарисовал особый вид линии (см. рис 7.1).



Рис. 7.1. – Линия Пеано

Для ее рисования Пеано использовал следующий алгоритм. На первом шаге он брал прямую линию и заменял ее на 9 отрезков длиной в 3 раза меньшей, чем длинна исходной линии (часть 1 и 2 рисунка 7.1). Далее он делал то же самое с каждым отрезком получившейся линии. И так до бесконечности. Ее уникальность в том, что она заполняет всю плоскость. Доказано, что для каждой точки на плоскости можно найти точку, принадлежащую линии Пеано. Кривая Пеано и пыль Кантора выходили за рамки обычных геометрических объектов. Они не имели четкой размерности. Пыль Кантора строилась вроде бы на основании одномерной прямой, но состояла из точек, а кривая Пеано строилась на основании одномерной линии, а в результате получалась плоскость. Во многих других областях науки появлялись задачи, решение которых приводило к странным результатам.

Вплоть до 20 века шло накопление данных о таких странных объектах, без какой либо попытки их систематизировать. Так было, пока за них не взялся Бенуа Мандельброт – отец современной фрактальной геометрии и слова фрактал. Работая в IBM математическим аналитиком, он изучал шумы в электронных схемах, которые невозможно было описать с помощью статистики. Постепенно сопоставив факты, он пришел к открытию нового направления в математике – фрактальной геометрии.

Из всех типов фракталов наиболее наглядными являются геометрические фракталы. В двухмерном случае их получают с помощью некоторой ломаной (или поверхности в трехмерном случае), называется генератором. За один шаг алгоритма каждый из отрезков, составляющих ломанную, заменяется на ломаную-генератор в соответствующем масштабе. В результате бесконечного повторения этой процедуры получается геометрический фрактал.

Одним из примеров таких фигур является звезда Кох. Для ее построения берут равносторонний треугольник и последовательно добавляют к нему новые, подобные ему, треугольники. На первом шаге стороны правильного треугольника (рис. 7.2) разбиваются на три равные части и их середины заменяются на правильные треугольники, подобные исходному. В результате получается правильный звездчатый шестиугольник (рис. 7.3). Стороны этого шестиугольника снова разбиваются на три равные части, и их середины заменяются на правильные треугольники (рис. 7.4). Повторяя этот процесс, получают все более сложные многоугольники (рис. 7.5, 7.6), все более приближающиеся к предельному положению.

Рассмотренная ранее кривая Пеано является геометрическим фракталом. Так же ниже приведены другие примеры геометрических фракталов (Лист, Треугольник Серпинского).

Вторая большая группа фракталов – алгебраические. Свое название они получили за то, что их строят, на основе алгебраических формул иногда весьма простых. Методов получения алгебраических фракталов несколько.

Один из методов представляет собой многократный (итерационный) расчет функции $Z_{n+1}=f(Z_n)$, где Z – комплексное число, а f – некая функция. Расчет данной функции продолжается до выполнения определенного условия. И когда это условие выполнится – на экран выводится точка.

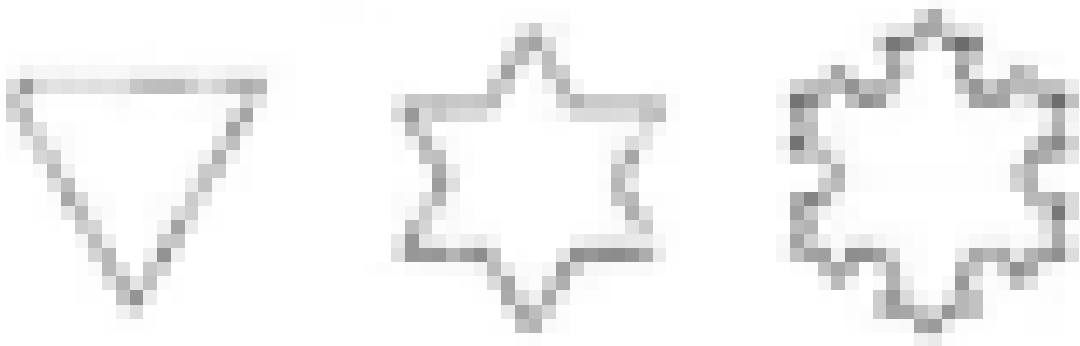


Рис. 7.2 – звезда Кох

Рис. 7.3 – звезда Кох

Рис. 7.4 – звезда Кох

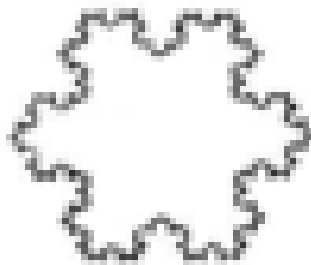


Рис. 7.5 – звезда Кох

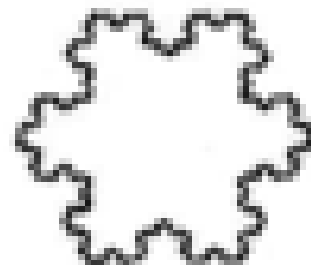


Рис. 7.6 – звезда Кох

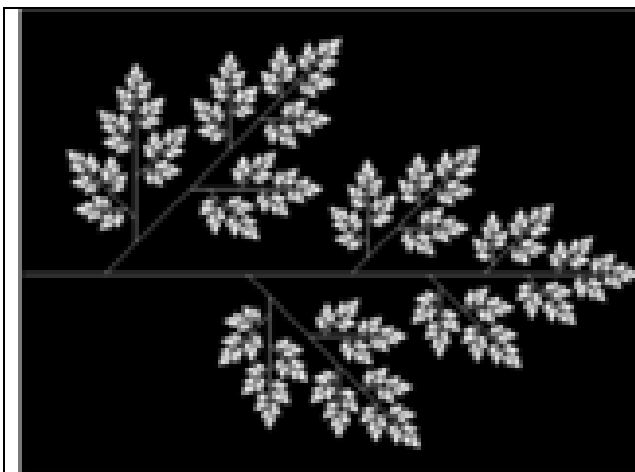


Рис. 7.7 – Лист

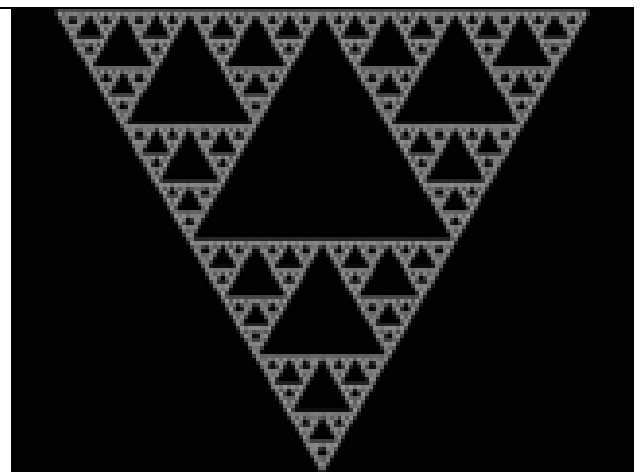


Рис. 7.8. – Треугольник Серпинского

Фракталом Мандельброта названа фигура, которая порождается очень простым циклом. Для создания этого фрактала необходимо для каждой точки изображения выполнить цикл итераций в соответствии с формулой:

$$z_{k+1} = z_k^2 + z_0, \quad (7.1)$$

где $k = 0, 1, \dots, n$. Величины z_k — это комплексные числа, $z_k = x_k + iy_k$, причем стартовые значения x_0 и y_0 — это координаты точки изображения. Для каждой точки изображения итерации выполняются ограниченное количество раз (n) или до тех пор, пока модуль числа z_k не превышает 2. Модуль комплексного числа равняется корню квадратному из $x^2 + y^2$.

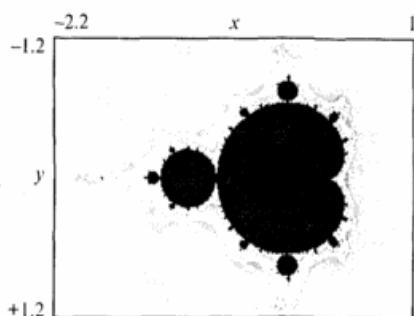


Рис. 7.9. – Фрактал Мандельброта

Для вычисления квадрата величины z_k можно воспользоваться формулой:

$$z^2 = (x + iy)(x + iy) = x^2 - y^2 + 2ixy, \quad (7.2)$$

поскольку $i^2 = -1$.

Цикл итераций для фрактала Мандельброта можно выполнять в диапазоне $x =$ (от -2.2, до 1), $y =$ (от -1.2 до 1.2).

Меняя функцию, условия выхода из цикла можно получать другие фракталы. Например, Фрактал Жулиа совсем не похож на фрактал Мандельброта, однако, он определяется итерационным циклом, почти полностью тождественным циклу генерации Мандельброта. Формула итераций для фрактала Жулиа такова:

$$z_{k+1} = z_k^2 + c, \quad (7.3)$$

где c — комплексная константа.

Условием завершения итераций является $|z_k| > 2$ — так же, как для фрактала Мандельброта.

Как видим, фрактал самоподобный (рис. 7.10) — при любом увеличении отдельные части напоминают формы целого. Самоподобие

считается важным свойством фракталов. Это отличает их от других типов объектов сложной формы.

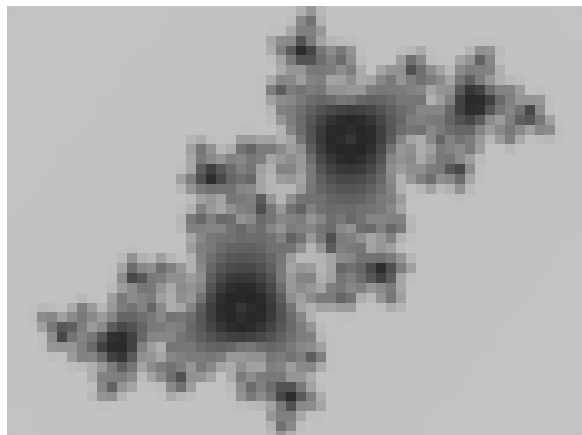


Рис. 7.10 – Фрактал Жулиа

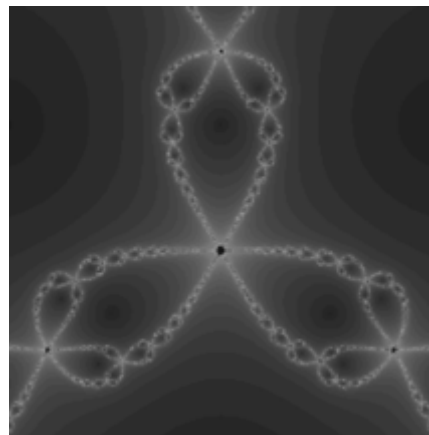


Рис. 7.11 – Фрактал Ньютон

Рассмотрим следующий пример фрактала — фрактал Ньютон. Для него итерационная формула имеет такой вид:

$$z_{k+1} = \frac{3z_k^4 + 1}{4z_k^3}, \quad (7.4)$$

где z — также комплексные числа, причем $z_0 = x + iy$ соответствует координатам точки изображения.

Условием прекращения цикла итераций для фрактала Ньютон есть приближение значений $|z^4 - 1|$ к нулю.

Еще одним известным классом фракталов являются стохастические фракталы, которые получаются в том случае, если в итерационном процессе случайным образом менять какие-либо его параметры. При этом получаются объекты очень похожие на природные — несимметричные деревья, изрезанные береговые линии и т.д. Двумерные стохастические фракталы используются при моделировании рельефа местности и поверхности моря

Типичный представитель данного класса фракталов «Плазма» (рис. 7.12).

Для ее построения возьмем прямоугольник и для каждого его угла определим цвет. Далее находим центральную точку прямоугольника и раскрашиваем ее в цвет равный среднему арифметическому цветов по углам прямоугольника плюс некоторое случайное число. Чем больше случайное

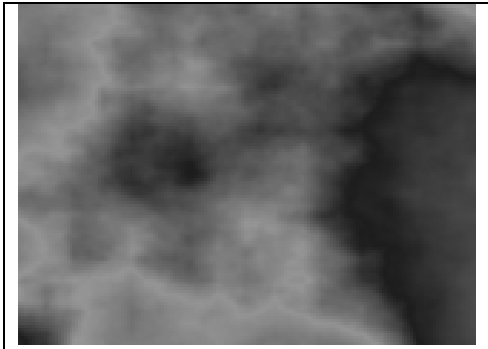


Рис.7.12 – Плазма

число – тем более «рваным» будет рисунок. Если, например, сказать, что цвет точки это высота над уровнем моря, то получим вместо плазмы – горный массив. Именно на этом принципе моделируются горы в большинстве программ. С помощью алгоритма, похожего на плазму строится карта высот, к ней

применяются различные фильтры, накладываем текстуру.

Пример построения фрактала.

Разработаем фрактал, который будет выглядеть как растение. Вообразим ствол, на котором много веточек. На каждой веточке много меньших веточек и так далее. Наименьшие ветви можно считать листвой или колючками. Все элементы будем рисовать отрезками прямой. Каждый отрезок будет определяться двумя конечными точками.

Для начала итераций необходимо задать стартовые координаты концов отрезка. Это будут точки 1, 2. На каждом шаге итераций будем рассчитывать координаты других точек.

Сначала находим точку 3. Это повернутая на угол α точка 2, центр поворота — в точке 1 (рис. 19):

$$\begin{aligned} x_3 &= (x_2 - x_1) \cos \alpha - (y_2 - y_1) \sin \alpha + x_1, \\ y_3 &= (x_2 - x_1) \sin \alpha + (y_2 - y_1) \cos \alpha + y_1. \end{aligned} \quad (7.5), (7.6)$$

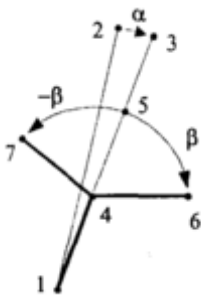


Рис. 7.13. Опорные точки элементов фрактала

Если $\alpha = 0$, то ствол и все ветви прямые. Потом находим точку 4. От нее будут распространяться ветви. Пусть соотношение длин отрезков 1-4 и 1-3 равняется k , причем $0 < k < 1$. Тогда для вычисления координат точки 4 можно воспользоваться такими формулами (7.7), (7.8).

$$\begin{aligned}x_4 &= x_1 (1 - k) + x_3 k, \\y_4 &= y_1 (1 - k) + y_3 k.\end{aligned}\tag{7.7), (7.8)}$$

Теперь зададим длину и угол наклона ветвей, которые растут из точки 4. Сначала найдем координаты точки 5. Введем еще один параметр — k_1 , который будет определять соотношение длин отрезков 4-5 и 4-3, причем $0 < k_1 < 1$. Координаты точки 5 равняются:

$$\begin{aligned}x_5 &= x_4 (1 - k_1) + x_3 k_1, \\y_5 &= y_4 (1 - k_1) + y_3 k_1.\end{aligned}\tag{7.9), (7.10)}$$

Точки 6 и 7 — это точка 5, но повернутая относительно точки 4 на углы β и $-\beta$ соответственно:

$$x_6 = (x_5 - x_4) \cos\beta - (y_5 - y_4) \sin\beta + x_4,\tag{7.11}$$

$$y_6 = (x_5 - x_4) \sin\beta + (y_5 - y_4) \cos\beta + y_4,\tag{7.12}$$

$$x_7 = (x_5 - x_4) \cos\beta + (y_5 - y_4) \sin\beta + x_4.\tag{7.13}$$

$$y_7 = -(x_5 - x_4) \sin\beta + (y_5 - y_4) \cos\beta + y_4,\tag{7.14}$$

Кроме расчета опорных точек, на каждом шаге будем рисовать один отрезок 1-4. В зависимости от номера итераций можно изменять цвет отрезка. Также можно устанавливать его толщину, например, пропорционально длине.

Таким образом, фрактал мы определили как последовательность аффинных преобразований координат точек. Величины a , β , k , k_1 — это параметры, которые описывают вид фрактала в целом. Они являются константами на протяжении всего итеративного процесса. Это дает возможность в итерациях использовать только операции сложения, вычитания и умножения, если вычислить значения $\sin()$, $\cos()$, $(1 - k)$ и $(1 - k_1)$ только один раз перед началом итераций как коэффициенты-константы.

Для того чтобы нарисовать фрактал, необходимо вызвать процедуру Draw, установив соответствующие значения ее аргументов: Draw(x_1 , y_1 , x_2 , y_2 , 0). Обратите внимание на один из аргументов этой процедуры — num, который сначала имеет значение 0. В теле процедуры есть три рекурсивных вызова с разными значениями этого аргумента:

- Draw(x4, y4, x3, y3, num)—продолжаем СТВОЛ;
- Draw(x4, y4, x6, y6, num+1)—правая ветвь;
- Draw(x4, y4, x7, y7, num+1) —левая ветвь.

Значение num показывает степень детализации расчета дерева. Один цикл итераций содержит много шагов, которые соответствуют одному значению величины num. Числовое значение num можно использовать для прекращения итеративного процесса, а также для определения текущего цвета элементов «растения».

Завершение циклов итераций в нашем алгоритме происходит тогда, когда длина ветви становится меньше некоторой величины lmin, например, 1.

Этот фрактал при $\alpha = 2^\circ$, $\beta = 86^\circ$, $k = 0.14$, $k_1 = 0.3$ похож на папоротник (рис. 7.14).

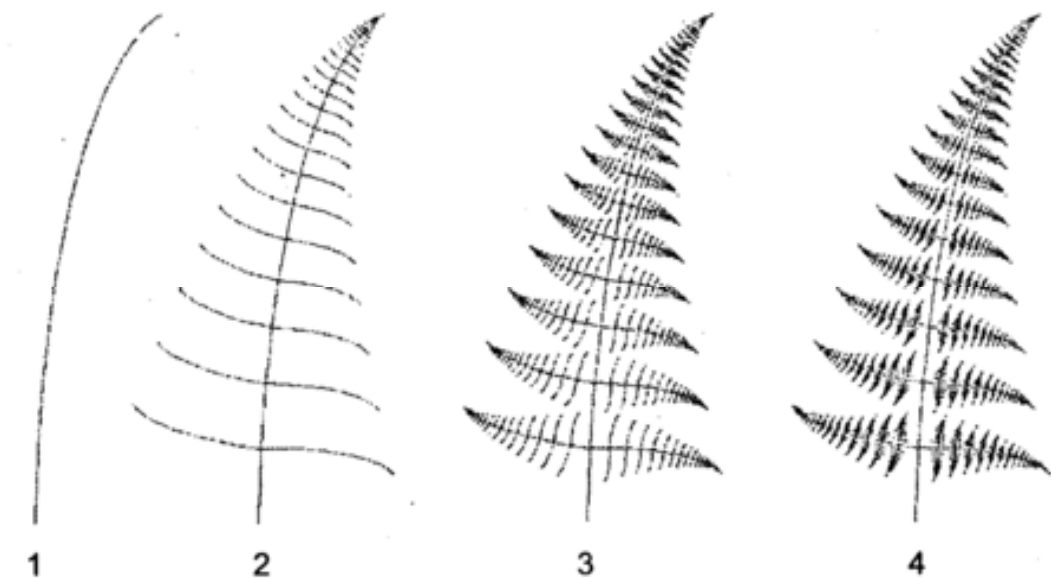


Рис. 7.14. – Вид фрактала для разного количества циклов итераций

```
void Draw(double x1, double y1, double x2 , double
y2, int num)
{
    int x3,x4,x5,x6,x7,y3,y4,y5,y6,y7;
    if (num >5) {return;}
    if( sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1))>L)
    {
        x3=floor((x2-x1)*cos(A)-(y2-y1)*sin(A)+x1);
        y3=floor((x2-x1)*sin(A)+(y2-y1)*cos(A)+y1);
```

```

x4=floor(x1*(1-k)+x3*k);
y4=floor(y1*(1-k)+y3*k);
x5=floor(x4*(1-k1)+x3*k1);
y5=floor(y4*(1-k1)+y3*k1);
x6=floor((x5-x4)*cos(B)-(y5-y4)*sin(B)+x4);
y6=floor((x5-x4)*sin(B)+(y5-y4)*cos(B)+y4);
x7=floor((x5-x4)*cos(B)+(y5-y4)*sin(B)+x4);
y7=floor(-(x5-x4)*sin(B)+(y5-y4)*cos(B)+y4);

line(x1,y1,x4,y4);

Draw(x4,y4,x3,y3,num);
Draw(x4,y4,x6,y6,num+1);
Draw(x4,y4,x7,y7,num+1);
}
}

```

Задание

Написать и отладить программу на языке C/C++ для рисования фрактала по варианту (табл. Ж.1, приложение Ж). Для этого:

1. Выполнить необходимые расчеты и реализовать функции для построения фрактала, заданного по варианту (табл. Ж.1., приложение Ж).
2. Получить графическое изображение фрактала.

Содержание отчета

1. Титульный лист.
2. Задание.
3. Разработка и описание метода получения фрактала.
4. Текст программы.

При защите лабораторной работы тестирование программы **обязательно!**

Лабораторная работа № 8.

Тема: поверхности.

Теоретические основы

Остановим внимание на двух широко распространенных трехмерных представлениях поверхностей в пространстве. При описании и конструировании внешних геометрических форм в системах автоматизированного проектирования, и других приложениях машинной геометрии и графики различают два подхода. Первый – основан на методах точного аналитического описания кривых и поверхностей, второй – использует приближенные методы: метод интерполяции и метод аппроксимации.

Аналитической моделью будем называть описание поверхности математическими формулами. Можно использовать много разновидностей такого описания. Поверхности, задаваемые аналитически, представляют в виде неявной функции $F(x, y, z) = 0$ или в явном виде, например, в виде функции двух аргументов $z = f(x, y)$ или параметрически.

Наиболее часто используется параметрическая форма описания поверхности. Запишем формулы для трехмерной декартовой системы координат (x, y, z) :

$$x = F_x(s, t), \quad (8.1)$$

$$y = F_y(s, t), \quad (8.2)$$

$$z = F_z(s, t), \quad (8.3)$$

где s и t — параметры, которые изменяются в определенном диапазоне, а функции F_x , F_y , и F_z определяют форму поверхности.

Преимущества параметрического описания — легко описывать поверхности, которые соответствуют неоднозначным функциям, замкнутые поверхности. Описание можно сделать таким образом, что формула не будет существенно изменяться при поворотах поверхности, масштабировании.

Иногда удобно поверхность отобразить как многогранник, используя формулы аналитического описания для расчета координат вершин граней в процессе отображения.

Полигональная сетка представляет собой совокупность ребер, вершин и многоугольников. Вершины соединяются ребрами, а многоугольники рассматриваются как последовательности ребер или вершин. Сетку можно представить несколькими различными способами, каждый из них имеет свои достоинства и недостатки.

Для описания пространственных объектов здесь используются такие элементы: вершины, отрезки прямых (векторы), полилинии, полигоны, полигональные поверхности (рис. 8.1).



Рисунок 8.1. Элементы векторно-полигональной модели

Элемент «вершина» (vertex) — главный элемент описания, все другие являются производными. При использовании трехмерной декартовой системы координат вершины определяются как (x, y, z) . Каждый объект однозначно определяется координатами собственных вершин.

Вершина может моделировать отдельный точечный объект, размер которого не имеет значения, а также может использоваться в качестве конечной точки для линейных объектов и полигонов. Двумя вершинами задается вектор.

Несколько векторов составляют полилинию. Полилиния может моделировать отдельный линейный объект, толщина которого не учитывается, а также может представлять собой контур полигона.

Полигон моделирует площадный объект. Один полигон может описывать плоскую грань объемного объекта. Несколько граней составляют объемный объект в виде полигональной поверхности — многогранник или

незамкнутую поверхность (в литературе часто употребляется название «полигональная сетка»).

Векторную полигональную модель можно считать наиболее распространенной в современных системах трехмерной компьютерной графики. Ее используют в системах автоматизированного проектирования, в компьютерных играх, в геоинформационных системах и т. п.

Пример. Построение сферы в виде многогранника, который аппроксимирует форму поверхности.

Уравнение сферы описывается формулами:

$x = a + R \cos \alpha \cos \nu,$	(8.4)
$y = b + R \cos \alpha \sin \nu,$	(8.5)
$z = c + R \sin \alpha.$	(8.6)

$a, b, c,$ – координаты центра.

```
void Draw(HWND hWnd)
{RECT rc;
HDC hdc;
point a,b,c,d;
POINT A[5];
int R=2, dl=10, db=10;
hdc=GetDC(hWnd);
if (hdc) {
GetClientRect(hWnd, &rc);
Rectangle(hdc, 0, 0, rc.right, rc.bottom);

for (int B=-90; B<=90; B+=db)
{
for (int L=0; L<=360; L+=dl)
{
a.x=R*cos(B*PI/180)*sin(L*PI/180); // 1 вершина
a.z=R*cos(B*PI/180)*cos(L*PI/180);
a.y=R*sin(B*PI/180);

b.x=R*cos((B+db)*PI/180)*sin(L*PI/180); //2
b.z=R*cos((B+db)*PI/180)*cos(L*PI/180);
b.y=R*sin((B+db)*PI/180);

c.x=R*cos((B+db)*PI/180)*sin((L+dl)*PI/180); //3
c.z=R*cos((B+db)*PI/180)*cos((L+dl)*PI/180);
c.y=R*sin((B+db)*PI/180);
```

```

d.x=R*cos(B*PI/180)*sin((L+d1)*PI/180); //4
d.z=R*cos(B*PI/180)*cos((L+d1)*PI/180);
d.y=R*sin(B*PI/180);
A[0]= MirToEk (rc.right,rc.bottom, a); //
перевод в экранные координаты
A[1]= MirToEk (rc.right,rc.bottom, b);
A[2]= MirToEk (rc.right,rc.bottom, c);
A[3]= MirToEk (rc.right,rc.bottom, d);
A[4]= MirToEk (rc.right,rc.bottom, a);
Polyline(hdc,A,5);
    }
}
}
ReleaseDC(hWnd,hdc);
}

```

Задание

Написать и отладить программу на языке C/C++ для рисования поверхности по варианту (табл. И.1, приложение И). Для этого:

1. Реализовать функции для построения поверхности, заданной по варианту (табл. И.1., приложение И) в виде многогранника, который аппроксимирует форму поверхности.
2. Получить графическое изображение поверхности.

Содержание отчета

5. Титульный лист.
6. Задание.
7. Разработка и описание метода получения поверхности.
8. Текст программы.

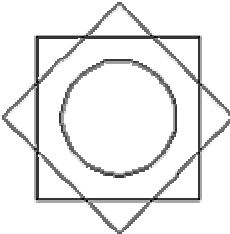
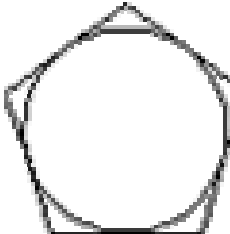
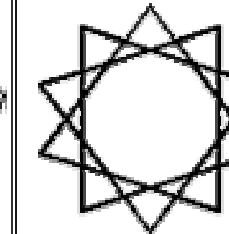
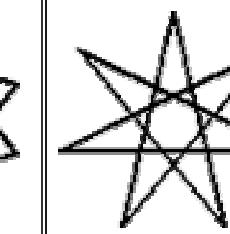
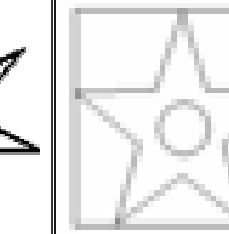
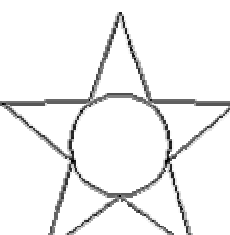
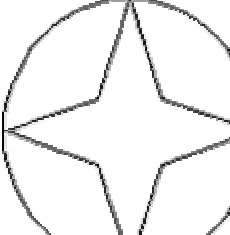
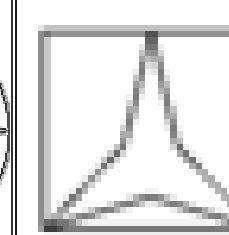
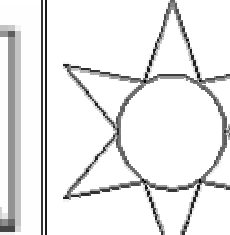
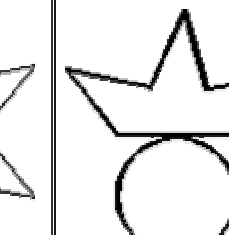
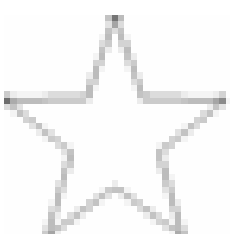
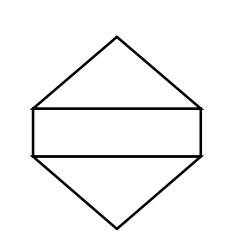
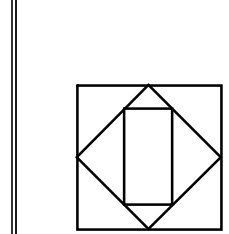
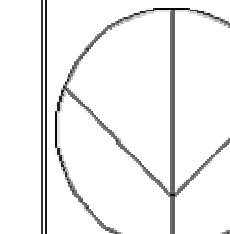
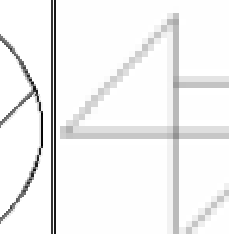
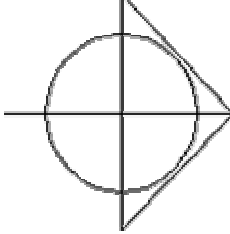
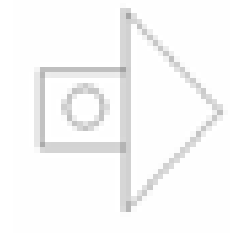
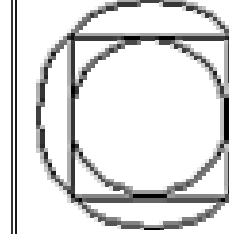
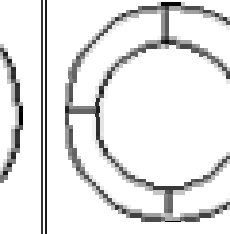
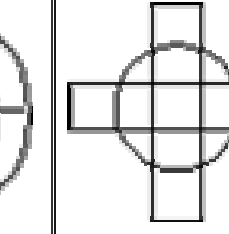
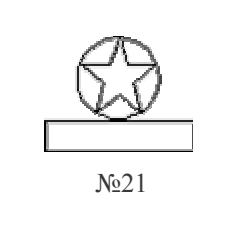
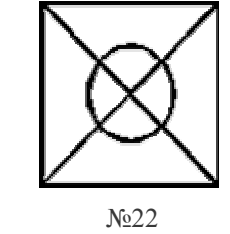
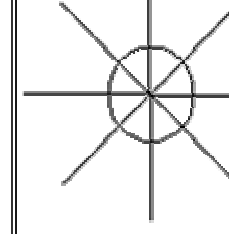
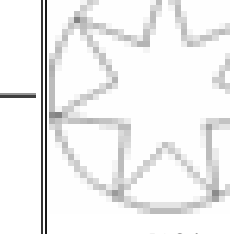
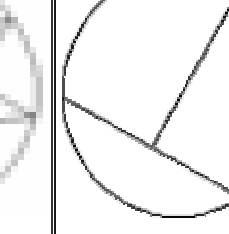
При защите лабораторной работы тестирование программы **обязательно!**

СПИСОК ЛИТЕРАТУРЫ

1. Компьютерная графика. / Блинова Т.А., Порев В.Н. – К.: Издательство Юниор, СПб.: КОРОНА принт, К.: Век+, 2006. – 520., ил.
2. Алгоритмические основы машинной графики./ Роджерс Д.: Пер. с англ. – М.: Мир, 1989. – 512 с., ил.
3. Математические основы машинной графики. / Роджерс Д., Адамс Дж.: Пер. с англ. – М.: Мир, 2001. —604с., ил.
4. Компьютерная графика. Математические основы. Версия 1.0 [Электронный ресурс]: электрон. учеб. пособие / Н. В. Соснин. – Электрон. дан. (4 Мб). Красноярск : ИПК СФУ, 2008.
5. Компьютерная графика и геометрическое моделирование. / Сидоренко Л. – Учебное пособие. Издательство Питер 2009. – 224с., ил.
6. Компьютерная геометрия и алгоритмы машинной графики. / Никулин Е.А. СПб.: БХВ-Петербург. – 2003. – 560с., ил.
7. Компьютерная графика. / Порев В.Н. – СПб.: БХВ-Петербург. – 2002. – 428с., ил.
8. Методы и алгоритмы компьютерной графики в примерах на Visual C++. / Поляков А. Ю., Брусенцев В. А. – 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2003. — 560 с: ил.
9. Как программировать на C++ / Дейтл Х.М. – Издательство Бином, 5-е полное издание – 2008. – 1454с., ил

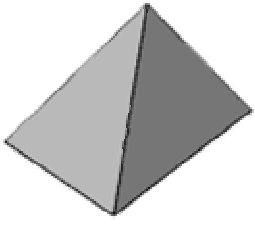
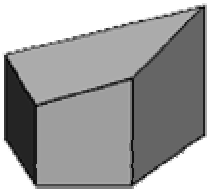
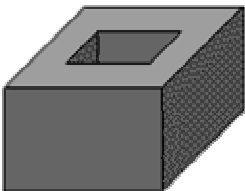
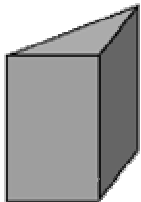
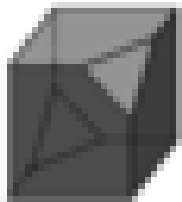
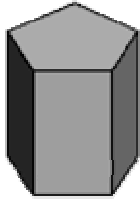
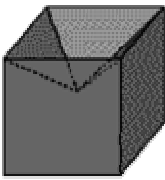
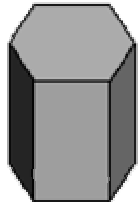
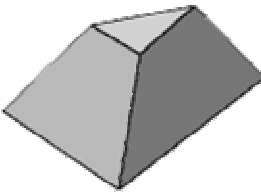
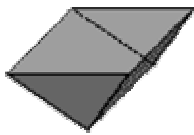
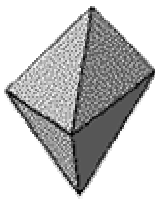
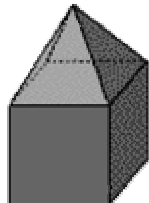
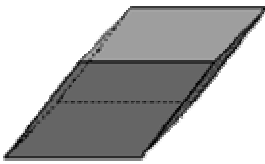
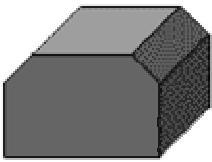
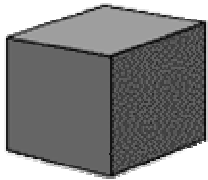
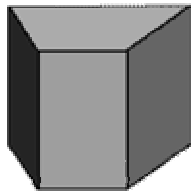
Графические фигуры

Таблица А.1. Варианты заданий

 №1	 №2	 №3	 №4	 №5
 №6	 №7	 №8	 №9	 №10
 №11	 №12	 №13	 №14	 №15
 №16	 №17	 №18	 №19	 №20
 №21	 №22	 №23	 №24	 №25




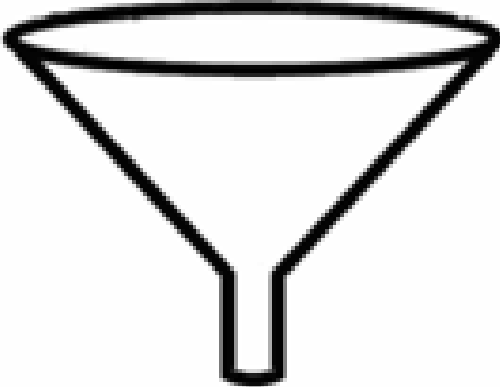

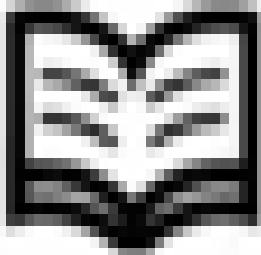
Пространственные тела

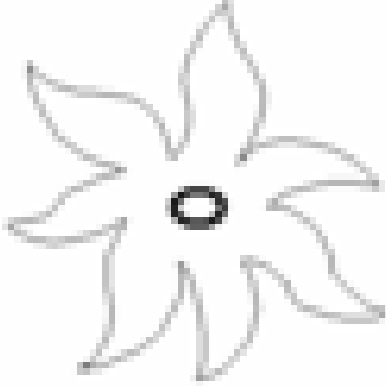

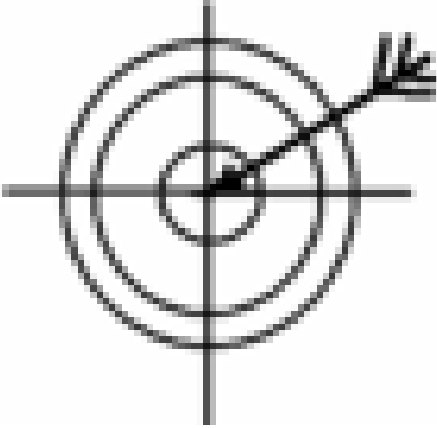


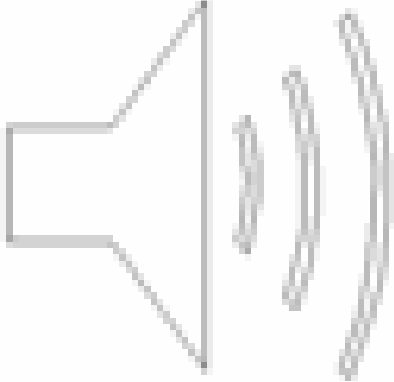
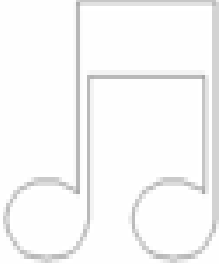

Таблица Б.1. Варианты заданий




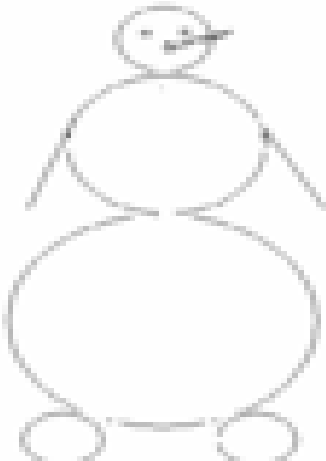
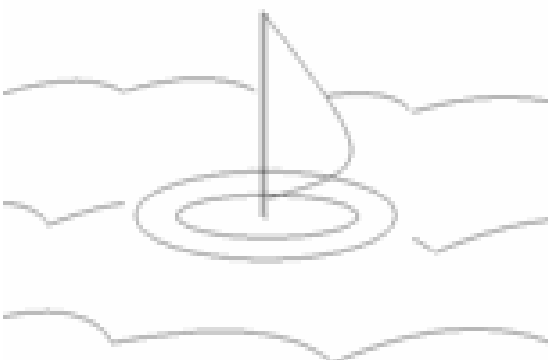
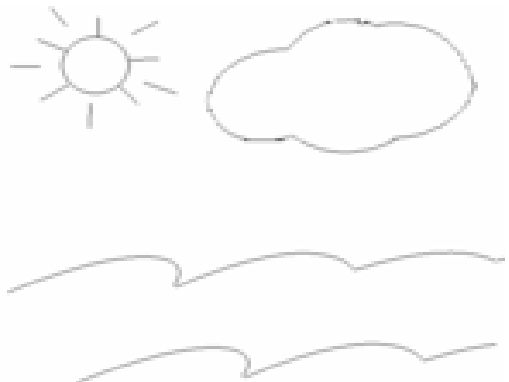

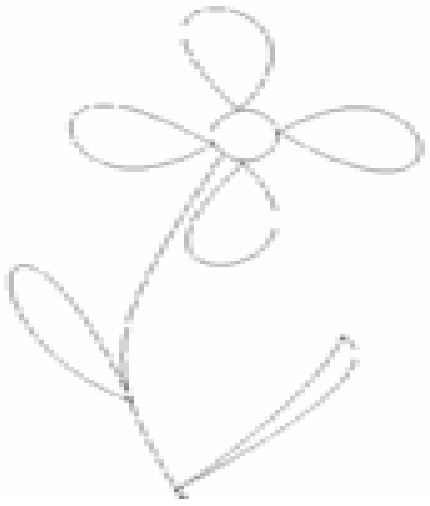
 <p>№1</p>	 <p>№2</p>	 <p>№3</p>	 <p>№4</p>
 <p>№5</p>	 <p>№6</p>	 <p>№7</p>	 <p>№8</p>
 <p>№9</p>	 <p>№10</p>	 <p>№11</p>	 <p>№12</p>
 <p>№13</p>	 <p>№14</p>	 <p>№15</p>	 <p>№16</p>

Графические фигуры

Таблица В.1. Варианты заданий

№ вар.	Рисунок	№ вар.	Рисунок
1		2	
3		4	
5		6	

7		8	
9		10	
11		12	
13		14	

15		16	
17		18	
19		20	
21		22	

Отсечение фигур

Таблица Д.1. Варианты заданий

№	Фигура	Движение	Цвет	Поле вывода	Цвет в зоне видимости
1	A(10,40), B(15,15), C(30,25), D(50,18), E(45,35).	диагональ → влево	красный	(100,100, 300, 300)	белый
2	A(10,40), B(15,15).	вниз → диагональ	серый	(200,200, 400, 400)	красный
3	A(10,50), B(15,25), C(30,35), D(50,28), E(45,45).	диагональ до центра экрана → влево	зеленый	(100,100, 300, 300)	красный
4	C(30,35), D(50,28).	диагональ до центра экрана → вниз	желтый	(200,200, 400, 400)	красный
5	A(5,40), B(10,15), C(15,10), D(45,18), E(40,35).	диагональ → вверх	красный	(100,100, 400, 400)	серый
6	D(45,18), E(40,35).	диагональ → влево	синий	(100,100, 300, 300)	зеленый
7	A(15,40), B(20,15), C(35,25), D(55,18), E(50,35).	диагональ до центра экрана → вниз	белый	(100,100, 400, 400)	синий
8	A(15,40), E(50,35).	диагональ → вверх до середины экрана	желтый	(200,200, 400, 400)	красный
9	A(15,45), B(20,20), C(35,30), D(55,25), E(50,40).	вниз → диагональ	серый	(100,100, 300, 300)	красный
10	B(20,20), D(55,25),	диагональ до центра экрана → вверх	желтый	(200,200, 400, 400)	красный
11	A(20,40), B(25,15), C(45,25), D(60,18), E(55,35).	вправо → диагональ	синий	(100,100, 300, 300)	зеленый
12	D(20,18), E(55,35).	диагональ до центра экрана → вниз	красный	(100,100, 400, 400)	серый
13	A(10,40), B(15,15), C(30,5), D(50,18), E(45,35).	вниз до середины экрана → вправо	серый	(100,100, 300, 300)	красный
14	D(50,18), E(45,35).	вниз → диагональ	синий	(100,100, 400, 400)	зеленый
15	A(20,49), B(25,5), C(40,35), D(60,40), E(70,65).	диагональ → вверх до середины экрана	желтый	(200,200, 400, 400)	красный
16	A(10,45), E(45,40).	диагональ до центра экрана → вниз	серый	(100,100, 300, 300)	красный
17	A(20,49), B(25,5), C(40,35), D(60,40), E(70,65).	диагональ до центра экрана → вправо	синий	(200,200, 400, 400)	зеленый
18	A(12,40), B(18,15).	диагональ → влево	серый	(100,100, 400, 400)	красный
19	A(12,40), B(18,15), C(35,25), D(54,18), E(47,35).	диагональ до центра экрана → вверх	зеленый	(100,100, 300, 300)	красный
20	D(54,18), E(47,35).	вниз → диагональ	синий	(200,200, 400, 400)	зеленый

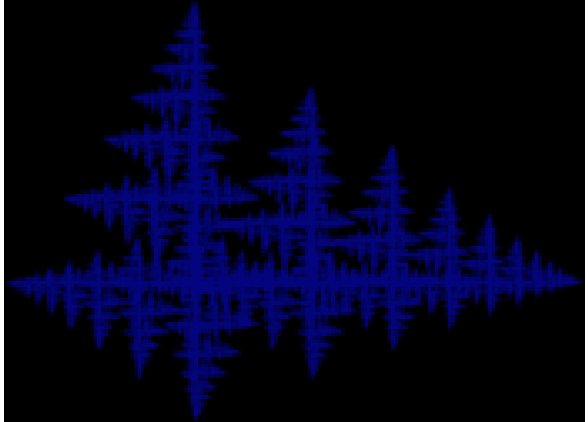
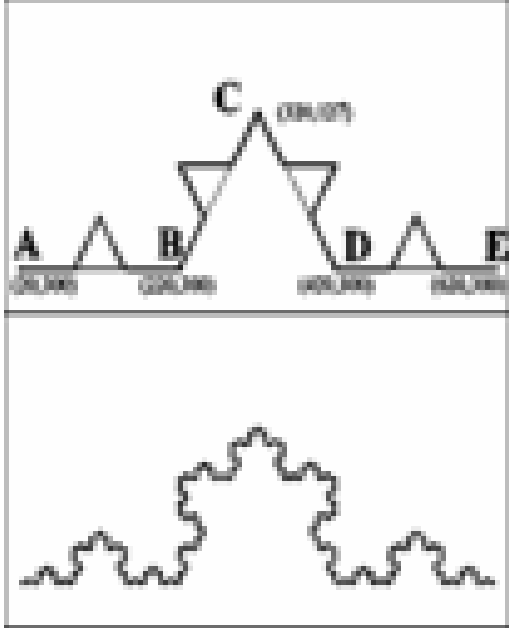

Заполнение фигур

Таблица Е.1. Варианты заданий

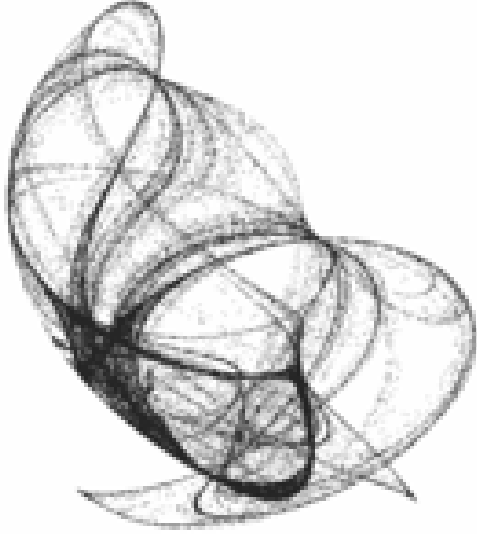
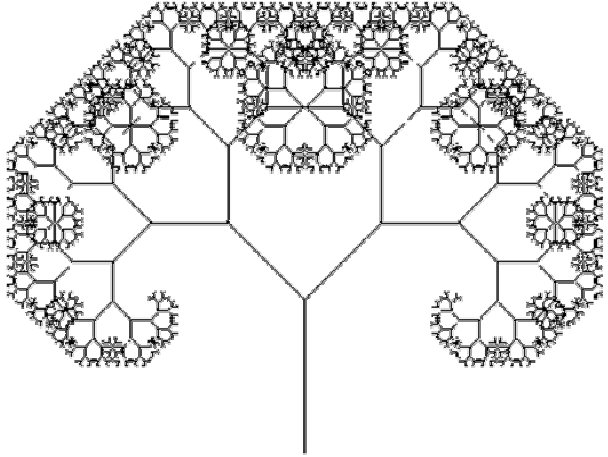
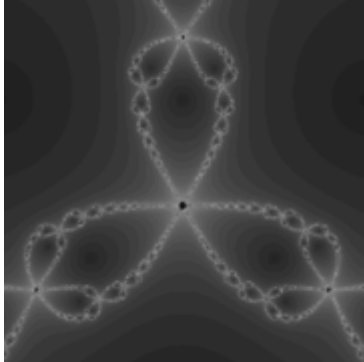

№	Фигура	Цвет	Окружность 1	Окружность 2	Цвет	Эллипс
1	A(10,40), B(15,15), C(30,25), D(50,18), E(45,35).	красный	X(350) Y(350) R=10 r=5	X(350) Y(50) R=15	синий	X(50) Y(350) R ₁ =15 R ₂ =5
2	A(20,40), B(25,15), C(45,25), D(60,18), E(55,35).	синий	X(450) Y(450) R=10 r=5	X(550) Y(50) R=25	красный	X(50) Y(450) R ₁ =15 R ₂ =5
3	A(10,50), B(15,25), C(30,35), D(50,28), E(45,45).	зеленый	X(500) Y(450) R=10 r=5	X(450) Y(50) R=15	белый	X(25) Y(350) R ₁ =15 R ₂ =5
4	A(10,40), B(15,15), C(30,5), D(50,18), E(45,35).	серый	X(550) Y(450) R=10 r=5	X(450) Y(80) R=50	желтый	X(20) Y(450) R ₁ =15 R ₂ =5
5	A(5,40), B(10,15), C(15,10), D(45,18), E(40,35).	красный	X(450) Y(450) R=10 r=5	X(450) Y(100) R=25	белый	X(75) Y(375) R ₁ =15 R ₂ =5
6	A(20,49), B(25,5), C(40,35), D(60,40), E(70,65).	желтый	X(550) Y(450) R=10 r=2	X(550) Y(80) R=15	зеленый	X(75) Y(475) R ₁ =15 R ₂ =5
7	A(15,40), B(20,15), C(35,25), D(55,18), E(50,35).	белый	X(375) Y(450) R=10 r=2	X(350) Y(100) R=20	красный	X(100) Y(350) R ₁ =15 R ₂ =5
8	A(10,45), B(15,20), C(30,30), D(50,25), E(45,40).	синий	X(350) Y(350) R=10 r=2	X(525) Y(80) R=50	желтый	X(100) Y(350) R ₁ =5 R ₂ =15
9	A(15,45), B(20,20), C(35,30), D(55,25), E(50,40).	серый	X(550) Y(450) R=10 r=3	X(550) Y(50) R=20	белый	X(50) Y(350) R ₁ =5 R ₂ =15
10	A(12,40), B(18,15), C(35,25), D(54,18), E(47,35).	зеленый	X(475) Y(450) R=10 r=2	X(450) Y(75) R=30	синий	X(150) Y(450) R ₁ =5 R ₂ =15

Фракталы

Таблица Ж.1. Варианты заданий

№	Название	Описание	Вид
1	фрактал Дендрит	<p>Данный фрактал строится благодаря случайному включению одной из двух пар уравнений для каждой итерации:</p> $x = a * x - b * y;$ $y = b * t + a * y;$ <p>или</p> $x = c * x - d * y + 1 - c;$ $y = d * t + c * y - d;$ <p>где a - наклон, b - коэффициент размера "листьев", c - "разброс" точек, d - искажение фрактала, t = x.</p>	
2	кривая Кох	<p>Эта кривая имеет четыре части, подобные целой кривой. Расположим первое поколение этого фрактала на сетке координат 640x350</p> <p>Для ее построения требуется набор аффинных преобразований, состоящий из четырех преобразований:</p> $X' = 0.333 * X + 13.333$ $Y' = 0.333 * Y + 200$ $X' = 0.333 * X + 413.333$ $Y' = 0.333 * Y + 200$ $X' = 0.167 * X + 0.289 * Y + 130$ $Y' = -0.289 * X + 0.167 * Y + 256$ $X' = 0.167 * X - 0.289 * Y + 403$ $Y' = 0.289 * X + 0.167 * Y + 71$ <p>Результат применения этого аффинного коллажа после десятой итерации можно увидеть на рисунке.</p>	
3	фрактал Мандельброта	<p>Для каждой точки изображения выполнить цикл итераций в соответствии с формулой:</p> $z_{k+1} = z_k^2 + z_0$ <p>где k = 0, 1, ..., n. Величины z_k — это комплексные числа, z_k = x_k + iy_k, причем стартовые значения x₀ и y₀ — это координаты точки изображения. Условие завершения итераций - z_k > 2</p>	

4	«дракон» Хартера-Хейтуэя.	<p>Для этого расположим первое поколение этого фрактала на сетке координат дисплея 640 x 350. Обозначим точки получившейся ломаной A,B,C. По правилам построения у этого фрактала две части, подобные целому это ломаные ADB и BEC. Зная координаты концов этих отрезков, можно вычислить коэффициенты двух аффинных преобразований, переводящих ломаную ABC в ADB и BEC:</p> $X' = -0.5 * X - 0.5 * Y + 490$ $Y' = 0.5 * X - 0.5 * Y + 120$ <p style="text-align: center;">и</p> $X' = 0.5 * X - 0.5 * Y + 340$ $Y' = 0.5 * X + 0.5 * Y - 110$ <p>Задавшись начальной стартовой точкой (напр. X=0 Y=0) и итерационно действуя на нее этой IFS, после десятой итерации на экране получим фрактальную структуру, которая представляет собой «дракон».</p>	
5	фрактал Martin.	<p>Фрактал строится благодаря многократному использованию следующих формул:</p> $x' = y - \sin(x)$ $y' = a - x,$ <p>x' и y' - значения, получаемые в результате подстановки в формулу значений, вычисленных на предыдущем шаге.</p>	
6	ковер Серпинского	<p>Алгоритм(рекурсивный):</p> <ol style="list-style-type: none"> 1.Рисуется квадрат стороны которого равны. 2.После этого каждая сторона квадрата делится на 3 части. 3.Затем из квадрата разделенного на 9 квадратов удаляется центральный квадрат. 4.Пункты 1-3 применяются к 8 оставшимся квадратам. 	

7	аттрактор де Йонга	<p>Для построения аттракторов де Йонга необходимо в цикле реализовывать следующие формулы :</p> $x' = \sin(A*y) - \cos(B*x)$ $y' = \sin(C*x) - \cos(D*y)$ <p>где A,B,C,D произвольные константы.</p> <p>Характерной чертой различных аттракторов является независимость от начальных значений координат, при их изменении множество точек, сгенерированных согласно данным уравнениям, будет тоже. И лишь меняя значения констант получим различные отображения.</p>	
8	дерево Пифагора	<p>Алгоритм:</p> <ol style="list-style-type: none"> 1) Строим вертикальный отрезок 2) Из верхнего конца этого отрезка рекурсивно строим еще 2 отрезка под определенными углами 3) Вызываем функцию построения двух последующих отрезков для каждой ветви дерева. <p>В классическом дереве Пифагора угол равен 45 градусам.</p>	
9	фрактал Ньютона	<p>Итерационная формула имеет такой вид:</p> $z_{k+1} = \frac{3z_k^4 + 1}{4z_k^3}$ <p>где z — комплексные числа, причем $z_0 = x + iy$ соответствует координатам точки изображения. Условием прекращения цикла приближение значений $x^4 - 1$ к нулю.</p>	
10	фрактал Жулиа	<p>Формула итераций для фрактала Жулиа такова:</p> $z_{k+1} = z_k^2 + c,$ <p>где c — комплексная константа. Условием завершения является $z_k > 2$</p>	

Поверхности

Таблица И.1. Варианты заданий

№	В виде неявной функции	Параметрическое представление	Описание
1	$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1.$	$x = a \cos \theta \sin \phi,$ $y = b \sin \theta \sin \phi,$ $z = c \cos \phi.$ $0 \leq \theta \leq 2\pi,$ $0 \leq \phi \leq 2\pi.$	<u>Эллипсоид.</u> Величины a, b, c – полуоси эллипсоида. Если все они различны, эллипсоид называется трехосным. В случае $a = b = c$ эллипсоид является сферой.
2	$\frac{x^2}{a^2} + \frac{y^2}{b^2} - \frac{z^2}{c^2} = 1.$	$x = a \cos \theta \operatorname{ch} \phi,$ $y = b \sin \theta \operatorname{ch} \phi,$ $z = c \operatorname{sh} \phi.$ $0 \leq \theta \leq 2\pi,$ $-\pi \leq \phi \leq \pi.$	<u>Однополостной гиперboloид.</u> в зависимости от соотношения параметров – трехосный ($a > b > c$), сжатый ($a = b > c$), вытянутый ($a = b < c$), равносторонний ($a = b = c$).
3	$\frac{x^2}{a^2} + \frac{y^2}{b^2} - \frac{z^2}{c^2} = -1.$	$x = a \operatorname{sh} \theta \operatorname{ch} \phi,$ $y = b \sin \theta \operatorname{ch} \phi,$ $z = c \cos \theta \operatorname{sh} \phi.$ $0 \leq \theta \leq 2\pi,$ $-\pi \leq \phi \leq \pi.$	<u>Двухполостный гиперboloид.</u> где a и b – мнимые полуоси; c – действительная полуось; в зависимости от соотношения параметров – трехосный ($a > b > c$), вращения ($a = b > c, a = b < c$), равносторонний ($a = b = c$).
4	$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$	$\begin{cases} x = u \sin \theta \cos v, \\ y = u \sin \theta \sin v, \\ z = H - u \cos \theta. \end{cases}$	<u>Конус.</u> В зависимости от значений полуосей a и b представляет собой эллиптический конус или круговой (при $a = b$). H – высота конуса; θ – угол между образующей и осью конуса; u, v – криволинейные координаты ($0 \leq u \leq H / \cos \theta, 0 \leq v \leq 360^\circ$).
5	$x^2 + y^2 = R$	$\begin{cases} x = R \cos v, \\ y = R \sin v, \\ z = u. \end{cases}$	<u>Прямой круговой цилиндр.</u> $0 \leq u \leq H, 0 \leq v \leq 360^\circ.$
6	$2z = \frac{x^2}{p} + \frac{y^2}{q}$	$x = a \phi \cos \theta,$ $y = b \phi \sin \theta,$ $z = \phi^2.$ $0 \leq \theta \leq 2\pi,$ $0 \leq \phi \leq \sqrt{2z}.$	<u>Эллиптический параболоид</u> (при положительных p и q).
8	$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1.$	$x = a \cos \theta, \quad 0 \leq \theta \leq 2\pi,$ $y = b \sin \theta, \quad \phi_{\min} \leq \phi \leq \phi_{\max},$ $z = \phi.$	<u>Эллиптический цилиндр.</u> Если $a = b$, то цилиндр оказывается круговым.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

По выполнению лабораторных работ

по курсу «КОМПЬЮТЕРНАЯ ГРАФИКА»

Для студентов специальности 6.050101 «Информационно управляющие системы и технологии» (ИУС).

Составители: Татьяна Александровна Васяева к.т.н., доц. каф. АСУ

Ольга Валентиновна Теплова ас. каф. АСУ

Формат 60x84, Ус. печ. лист. _____

Тираж – электронный вариант

83000, м. Донецк, ул. Артема 58, ДонНТУ.