**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по лабораторной работе №3**

**по дисциплине «Организация ЭВМ и Систем»**

Тема: **Исследование видеосистемы (графический режим)**

|  |  |  |
| --- | --- | --- |
| Студент(ка) гр. |  | Алексеев Г. |
| Преподаватель |  | Гречухин М. Н. |

Санкт-Петербург

2023

**Краткие сведения о видеосистемах ПЭВМ, графическом режиме их работы и функциях обслуживания графического режима.**

Графические режимы, поддерживаемые библиотекой графики, задаются символическими константами, описанными в заголовочном файле <graphics.h> в перечислимом типе graphics\_modes. Инициализацию графической модели выполняет функция initgraph(). При вызове она инициализирует графическую систему, загружая .BGI-драйвер, определяемый указателем graphdriver, и устанавливая видеоадаптер в графический режим, задаваемый указателем graphmode. Аргумент pathtodriver указывает на ASCII-строку, хранящую спецификацию файла .BGI-драйвера. С++ поддерживает фиксированное число драйверов, каждый из которых, в свою очередь, поддерживает ряд режимов. Как тип драйвера, так и режим могут быть заданы числом или символической константой. Третий аргумент функции initgraph() задает маршрут поиска файла, содержащего .BGI-драйвер. Если файл не найден в заданной директории, функция просматривает текущий директорий. Если pathtodriver = NULL, драйвер должен располагаться в текущей директории. В случае, когда при вызове initgraph() параметры видеосистемы неизвестны, значение для graphdriver следует задать равным указателю на DETECT. Благодаря этому функция initgraph() вызывает другую библиотечную функцию – detectgraph() - для определения типа видеоадаптера, подходящего графического драйвера и графического режима максимального разрешения (максимального режима) для активного видеоадаптера системы. Значения для драйвера и максимального режима возвращаются в ячейках памяти, на которые указывают graphdriver и graphmode. Если функции графической библиотеки больше не нужны прикладной программе, следует вызвать функцию closegraph() "закрытия" графического режима и возвращения к текстовому режиму.

Целая группа функций – getgraphmode(), getmaxmode(), getmodename() , getmoderange() - упрощает работу по определению текущего установленного режима. Две функции позволяют определить ширину и высоту экрана в пикселах для текущего видеорежима: getmaxx() и getmaxy(). Функция restorecrtmode() возвращает видеоадаптер в текстовый режим.

int getgraphmode (void) возвращает текущий графический режим (возвращаемое число соответствует номеру режима)

int getmaxmode(void) возвращает число, определяющее максимально возможный для инсталлированного (возращаемое число соответствует номеру режима)

int getmaxx(void) и int getmaxy(void) возвращают максимальные значения координат X и Y для текущего видеорежима.

char \* getmodename(int mode\_number) возвращает указатель на ASCII-строку символов, содержащую имя символической константы, соответствующей режиму mode\_number.

void setgraphmode(int mode) устанавливает видеосистему в режим, заданный значением переменной mode, и сбрасывает значения внутренних переменных системы графики в их значения по умолчанию

void restorecrtmode(void) возвращает видеоадаптер в режим, в котором он был до выполнения инициализации системы графики (текстовый).

Возможности по выбору цветов принципиально различны для CGA-, EGA- и VGA-адаптеров, что обусловлено различной логикой построения аппаратных средств.

int getbkcolor(void) возвращает целое число, равное коду цвета фона.

int getmaxcolor(void) возвращает максимальное значение кода цвета пиксела минус 1 (устанавливает максимальное число цветов).

void setbkcolor (int color) устанавливает новый цвет пикселов, имеющих код цвета 0. Новый цвет фона задает значение аргумента color.

void setcolor (int color) устанавливает цвет, используемый функциями графического вывода в значение, заданное аргументом color.

Окно экрана в графическом режиме, или графическое окно (viewport), - это прямоугольная область экрана, заданная пиксельными координатами левого верхнего и правого нижнего углов. Для описания окна используется функция setviewport().Текущие характеристики окна доступны программе через обращение к функции getviewsettings().

void far getviewsettings( struct viewporttype \*viewport) заполняет поля структурной переменной по шаблону viewporttype информацией о графическом окне.

void setviewport (int left, int top, int right, int bottom, int clip) Описывает новое графическое окно с координатами (столбец, строка) левого верхнего угла left, top, координатами правого нижнего угла right, bottom и значением флага усечения clip.

int getx (void) и int gety (void) возвращают текущие координаты X и Y, измеряемые относительно координат левого верхнего угла текущего графического окна.

void moveto (int x, int y) устанавливает новое значение координат текущей позиции.

void moverel(int dx, int dy) устанавливает новое значение координат текущей позиции, аргументы dx, dy задают новые значения координат относительно текущих координат графического окна.

Библиотека графики позволяет выводить на экран текст различными шрифтами. С++ имеет два типа шрифтов: битовый и сегментированный. Каждому символу битового шрифта (bit-mapped font) ставится в соответствие матрица пикселов фиксированного размера. Другой тип шрифтов, используемый при выводе текста на экран, фактически задает правило "рисования" каждого символа. Он описывается как совокупность отрезков прямых линий, или сегментов. Программа может задать масштаб для каждого символа, "растягивая" или "сжимая" его по высоте либо ширине. Однако использование сегментированного шрифта для вывода текста несколько замедляет работу видеосистемы. В С++ доступны четыре сегментированных шрифта: Triplex, Small, Sans-Serif и Gothic. Поведение системы графики при выводе текста в графическом режиме задается целой группой значений внутренних переменных. Их текущие установки доступны после вызова функции gettextsettings().

void gettextsettings( struct textsettingstype \*texttypeinfo) заполняет поля структурной переменной по шаблону textsettingstype информацией о текущих шрифте, направлении вывода текста, размере знакоместа относительно шрифта по умолчанию и способе "прижатия" (выравнивания) шрифта в пределах знакоместа.

void settextstyle(int font, int direction, int charsize) выбирает шрифт, устанавливает направление и размер знакоместа для последующего вывода текстовой информации через функции библиотеки графики outtext() и outtextxy(). Значение font выбирает один из шрифтов Turbo С. Возможные типы шрифтов задаются либо целым числом, либо символической константой из перечислимого типа font\_names. Значение direction позволяет специфицировать направление вывода. Если direction = HORIZ\_DIR, текст будет выводиться горизонтально слева направо. Если direction = VERT\_DIR, текст будет выводиться вертикально снизу-вверх, а символы будут повернуты на 90 градусов против хода часовой стрелки. charsize - задает масштаб каждого символа относительно знакоместа 8x8.

int textheight(char \*textstring) возвращает высоту строки символов в пикселах, на которую указывает textstring.

int textwidth (char far \*textstring) возвращает ширину строки символов в пикселах, на которую указывает textstring.

textwidth("A") возвращает ширину символа 'А' в пикселах.

void far settextjustify(int horiz,int vert) задает новую установку выравнивания символов текста в графических режимах работы адаптера. Аргумент horiz может принимать три значения, задаваемых символическими константами: LEFT\_TEXT - левая граница строки "прижимается" справа к вертикальной линии, проведенной через точку отсчета; CENTER\_TEXT - строка располагается так, что вертикальная линия, проведенная через ее середину, проходит через точку отсчета; RIGHT\_ТЕХТ - правая граница строки "прижимается" слева к вертикальной линии, проведенной через точку отсчета. Аргумент vert также может принимать три значения, задаваемых символическими константами: ВОТТОМ\_ТЕХТ - нижняя граница строки "прижимается" сверху к горизонтальной линии, проведенной через точку отсчета; CENTER\_TEXT - строка располагается так, что горизонтальная линия, проведенная через ее середину, проходит через точку отсчета; ТОР\_ТЕХТ - верхняя граница строки "прижимается" снизу к горизонтальной линии, проведенной через точку отсчета.

void outtext (char \*textstring) выводит ASCII-строку текста, на начало которой указывает textstring, используя текущие позицию, цвет и установки направления, типа шрифта и выравнивания строки.

void outtextxy (int x, int y, char \*textstring) выводит ASCII-строку текста, на начало которой указывает textstring, используя текущие цвет, установки направления, типа шрифта и выравнивания строки.

Для графического вывода используется текущий цвет пиксела, установленный функцией setcolor(). Для определения текущей установки стиля используется функция getlinesettings(). Выбор подходящего стиля выполняет функция setlinestyle().

void getlinesettings (struct linesettingstype \*lineinfo) возвращает информацию об установленном в текущий момент времени стиле "рисования" отрезков прямых линий и графических примитивов.

void setlinestyle (int linestyle, unsigned upattern, int thickness) устанавливает стиль "рисования" отрезков прямых линий и графических примитивов. Аргумент linestyle выбирает стиль линии, а аргумент thickness - толщину линии. Аргумент upattern используется только в том случае, когда задается отличный от предопределенных стиль линии, т.е. если linestyle равен USERBIT\_LINE (4).

void setwritemode(int mode) устанавливает режим вывода отрезков прямых линий в значение, определяемое аргументом mode. Аргумент mode может принимать одно из двух значений, описанных в <graphics.h>: COPY\_PUT (0) - пикселы, лежащие на отрезке прямой линии, переопределяют пикселы на экране, и, таким образом, линия на экране имеет текущий цвет; XOR\_PUT (1) - пикселы, образующие линию, имеют код цвета, образуемый операцией исключающего ИЛИ (XOR) кода текущего цвета и кода цвета пикселов на экране, через которые линия проходит. В частности, можно стереть выведенную линию с экрана, выполнив вывод линии еще раз.

void getaspectratio (int \*xasp, int \*yasp) заполняет две переменные, описанные точкой вызова, значениями коэффициента сжатия для текущего видеорежима.

void setaspectratio (int xasp, int yasp) устанавливает новое значение коэффициента сжатия, которое будет использоваться системой графики при выводе геометрических примитивов (прямоугольников и т.п.).

setaspectratio(100,120) корректировка вывода графической информации при использовании нестандартных мониторов, для которых не может автоматически определить корректное значение коэффициента сжатия, а также корректировка графического вывода для мониторов с некорректной линейностью по вертикали и горизонтали.

void getfillpattern (char \* pattern) заполняет область памяти из 8 байт, описанную точкой вызова, текущим значением маски заполнения. Аргумент pattern указывает на начало описанной области памяти

void setfillpattern(char \*upattern, int color) задает цвет пикселов и маску для заполнения областей экрана.

void getfillsettings( struct fillsettingstype \*fillinfo) заполняет поля структурной переменной по шаблону struct fillsettingstype информацией о текущей маске и цвете заполнения.

void setfillstyle(int pattern, int color) выбирает один из предопределенных стилей заполнения. Значение pattern идентифицирует стиль.

unsigned getpixel(int x, int у) определяет, лежит ли пиксел с координатами (х, у) в текущем графическом окне, и, если лежит, возвращает код цвета этого пиксела, иначе возвращается 0.

void putpixel(int x, int у, int pixelcolor) определяет, лежит ли пиксел с координатами (х, у) в текущем графическом окне, и, если лежит, выводит на экран пиксел, код цвета которого равен pixelcolor, иначе цвет пиксела не изменяется.

Используя функцию putpixel(), можно "стереть" пиксел, если вывести его с кодом цвета фона.

void arc(int x, int y, int stangle, int endangle, int radius) выводит дугу окружности радиусом radius. Центр окружности задают координаты х, у. Аргументы stangle и endangle задают соответственно начальный и конечный углы дуги. Углы задаются в градусах и отсчитываются против хода часовой стрелки.

void bar(int left, int top, int right, int bottom) выводит полосу, заполненную текущим цветом с использованием текущей маски заполнения.

void bar3d(int left, int top, int right, int bottom, int depth, int topflag) выводит в изометрии "столбик" и заполняет его фронтальную поверхность текущим цветом с использованием текущей маски заполнения. Аргументы задают: пиксельные координаты левого верхнего (left, top) и правого нижнего (right, bottom) углов заполняемой области экрана; "глубину "(depth) в пикселах изображаемого столбца.

void circle int x, int y, int radius) выводит окружность заданного аргументом radius радиуса с центром, заданным координатами х и у. Координаты центра определяются относительно координат левого верхнего угла текущего графического окна.

Хотя окружность может быть выведена и функцией агс(), использование circle() для этих целей предпочтительнее, так как для полной окружности эта функция более производительная.

void drawpoly(int numpoints, int polypoints[]) "соединяет" отрезками прямых линий текущего цвета и стиля точки (полигон), координаты которых заданы парами значений. Эти пары расположены в массиве, на который указывает polypoints[] . Аргумент numpoints задает число соединяемых между собой точек. Координаты точек задаются относительно координат левого верхнего угла текущего графического окна.

void ellipse (int x, int y, int stangle, int endangle, int xradius, int yradius) выводит эллиптическую дугу или полный эллипс, используя текущий цвет. Аргументы задают: пиксельные координаты центра эллипса (х, у); начальный угол дуги (stangle); конечный угол дуги (endangle); радиус эллипса по горизонтали (xradius); радиус эллипса по вертикали (yradius).

void fillellipse(int x, int y, int xradius, int yradius) выводит эллипс, заполненный текущим стилем. Аргументы функции задают: пиксельные координаты центра эллипса (х, у); радиус эллипса по горизонтали (xradius); радиус эллипса по вертикали (yradius). Функция автоматически корректирует координаты точек в соответствии с коэффициентом сжатия дисплея.

void fillpoly(int numpoints, int \*polypoints) выводит контур полигона, заданного numpoints точками. Координаты точек заданы парами, расположенными в массиве, на который ссылается polypoints.

void floodfill (int x, int y, int border) аполняет текущим стилем область экрана, ограниченную непрерывной линией с цветом border, начиная с точки с координатами (х, у). Функция заполняет область либо внутри замкнутой линии, либо вне ее (зависит от расположения точки).

void pieslice(int x, int y, int stangle, int endangle, int radius) выводит контур кругового сектора и заполняет его внутреннюю область текущим стилем.

void rectangle(int left, int top, int right, int bottom) выводит контур прямоугольника, заданного координатами левого верхнего (left, top) и правого нижнего (right, bottom) углов.

void sector(int x, int y, int stangle, int endangle, int xradius, int yradius) работает аналогично функции pieslice(), за исключением того, что выводится не круговая, а эллиптическая дуга.

**Цель работы.**

Изучение работы с видеосистемой в графическом режиме, вывод графика заданной функции с масштабированием и разметкой осей.

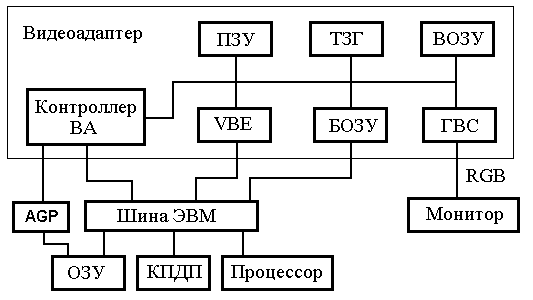
**Задание на лабораторную работу.**

1. Разработать программу для вывода на экран графика заданной функции

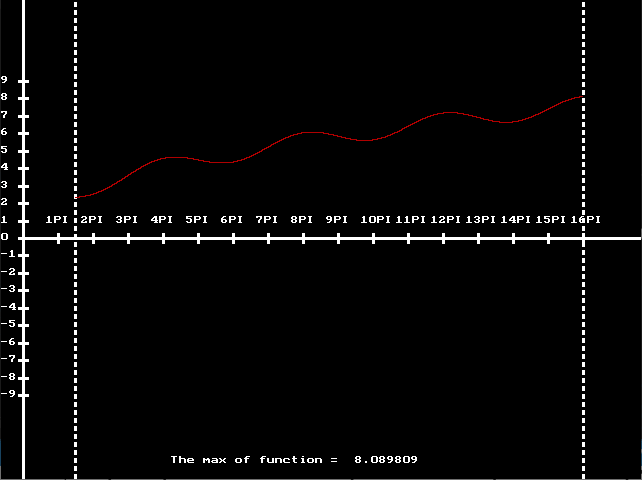


1. Произвести разметку осей и проставить истинные значения точек.
2. Найти максимальное значение функции на заданном интервале и вывести в отдельное окно на экране.

**Структурная схема аппаратных средств**



**Результат работы программы**



**Выводы.**

В результате выполнения работы были изучены принципы работы видеосистемы в графическом режиме и получены навыки по работе с ней. Написание и отладка программы происходили в Turbo C++.

**Исходный код программы**

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <dos.h>

#include <graphics.h>

#include <math.h>

int main(){

const double pi = 3.14159265358979323846;

const double x\_approach = 35/pi; // approximation coefficient

const double y\_approach = 17.5; // approximation coefficient

const int ten\_num = 10;

const float start\_out = 1.5; // start pi coefficient

const int end\_out = 16; // end pi coefficient

const int hatch\_out = 35;

const int number\_out = 10;

const int lines\_out = 22;

double y\_result; // y coordinate

double x\_result; // x coordinate

double y\_out; // y coordinate for output

double x\_out; // x coordinate for output

double max\_result = -100; // function maximum

int i;

int gd, gm;

int max\_x; // maximal x coodrinate in window

int max\_y; // maximal y coodrinate in window

char symbols\_out[4];

char maximum\_out[50];

clrscr();

gd = DETECT;

detectgraph(&gd, &gm);

initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

// GETTING MAXIMAL NUMBER OF X AND Y COORINATES IN VIDEOMODE

max\_x = getmaxx();

max\_y = getmaxy();

setviewport(0, 0, max\_x, max\_y, 0);

setlinestyle(0, 0, 3);

line(lines\_out, max\_y, lines\_out, 0);

line(lines\_out, max\_y/2, max\_x, max\_y/2);

for(i = start\_out; i < end\_out + 1; i++){

sprintf(symbols\_out, "%dPI", i);

outtextxy(ten\_num + hatch\_out\*i, max\_y/2 - lines\_out, symbols\_out);

line(lines\_out + hatch\_out\*i, max\_y/2 + ten\_num/2, lines\_out + hatch\_out\*i, max\_y/2 - ten\_num/2);

}

for(i = -number\_out + 1; i < number\_out; i++){

sprintf(symbols\_out, "%d", i);

outtextxy(0, max\_y/2 - hatch\_out\*i/2 - ten\_num/2, symbols\_out);

line(lines\_out + ten\_num/2, max\_y/2 - hatch\_out\*i/2, lines\_out - ten\_num/2, max\_y/2 - hatch\_out\*i/2);

}

//the start assymptote

setlinestyle(3, 0, 3);

line(lines\_out + hatch\_out\*start\_out, max\_y, lines\_out + hatch\_out\*start\_out, 0);

setlinestyle(0, 0, 3);

setviewport(0, 0, max\_x, max\_y, 0);

// the graph of function

for(x\_result = start\_out\*pi; x\_result <= end\_out\*pi; x\_result = x\_result + 0.001){

y\_result = pow(cos(x\_result/4), 2) + sqrt(x\_result);

x\_out = x\_result\*x\_approach;

y\_out = y\_result\*y\_approach;

if(y\_result > max\_result){

max\_result = y\_result;

}

putpixel(lines\_out + x\_out, max\_y/2 - y\_out, 4);

}

// the end assymptote

setlinestyle(3, 0, 3);

line(lines\_out + hatch\_out\*end\_out, max\_y, lines\_out + hatch\_out\*end\_out, 0);

// output the max of function

sprintf(maximum\_out, "The max of function = %f", max\_result);

outtextxy(max\_x/3, max\_y - lines\_out, maximum\_out);

getch();

closegraph();

return 0;

}