МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра ВТ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Организация ЭВМ и систем»

Тема: Исследование видеосистемы (графический режим)

Студентки гр. 3372 _____ Козина П.С.
Шорсткая А.А.

Преподаватель _____ Кочетков А.В

Санкт-Петербург
2024

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

Использование графики в языке C++ — это многошаговый процесс. Прежде всего необходимо определить тип видеоадаптера. Затем устанавливается подходящий режим его работы и выполняется инициализация графической системы в выбранном режиме. После этого становятся доступными для использования функции графической библиотеки graphicx.h для построения основных графических примитивов: отрезков прямых линий, окружностей, эллипсов, прямоугольников, секторов, дуг и т.д., появляется возможность вывода текста с использованием различных шрифтов.

Использование библиотеки графики намного сокращает объем программирования для вывода основных графических примитивов. C++ "маскирует" многие технические детали управления оборудованием, о которых пользователь должен быть осведомлен при работе с видеоадаптером через порты или BIOS. Платой за эти удобства является значительное увеличение размера EXE-файлов. Использование графической библиотеки C++ требует знакомства с моделью графической системы, применяемой компилятором для представления графической системы компьютера. Можно сказать, что сложность овладения деталями аппаратных средств видеоадаптеров сравнима со сложностью освоения графической модели. Однако достоинство графической модели заключается в ее относительной независимости от различных типов видеоадаптеров и открытости для дальнейших расширений. Появление новых типов видеоадаптеров не потребует большой переработки программ, так как все новые особенности аппаратуры будут учитываться в средствах библиотеки C++.

Весь код библиотеки графики разбивается на две части: немобильную, которая зависит от типа видеоадаптера и мобильную.

Немобильная часть представляет собой так называемый BGI-драйвер (BGI - Borland Graphics Interface). Драйвер является обработчиком прерывания 10h, который должен дополнить системный обработчик до того, как будут использоваться мобильные функции. Перед завершением программы таблица векторов прерывания восстанавливается.

Основные функции, выполняемые .BGI-драйвером, сводятся к установке и обновлению ряда внешних переменных, которые могут изменяться как функциями системного обработчика прерывания 10h (например, при переключении видеорежима, изменении регистров палитры и т.п.), так и мобильными функциями библиотеки графики. C++ включает целую коллекцию драйверов для каждого из типов адаптеров, хранимых обычно в отдельном поддиректории. Система графики является открытой для расширений, так как позволяет использовать и собственные .BGI-драйверы. Сложность состоит в

том, что фирма Borland International не раскрывает пока внутреннюю структуру драйвера. Совокупность внешних переменных библиотеки графики и особенностей поведения мобильных функций образует модель графики C++.

Прежде чем использовать функции графической библиотеки C++, необходимо инициализировать систему графики - загрузить соответствующий адаптеру или режиму .BGI-драйвер, установить в начальные значения внешние переменные и константы, выбрать шрифт и т.д.

Графические режимы, поддерживаемые библиотекой графики, задаются символическими константами, описанными в заголовочном файле <graphics.h> в перечислимом типе graphics_modes. Константы, определяющие видеорежим, приведены в табл. 3.1 вместе с информацией о выбираемом режиме и типе видеоадаптера, который может такой режим поддерживать.

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. C++ 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. Возможные значения для графических режимов даны в табл. 3.1. В табл. 3.2. приведены значения, определяющие графические драйверы при инициализации системы графики. Упомянутые в таблице символические константы определены в перечислимом типе graphics_drivers из заголовочного файла <graphics.h>.

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.

Помимо перевода видеоадаптера в заданный графический режим, функция initgraph() динамически распределяет оперативную память для загружаемого драйвера и хранения промежуточных результатов, возникающих

при работе некоторых функций графики. После загрузки драйвера initgraph() устанавливает в значения по умолчанию ряд параметров графики: стиль линий, шаблоны заполнения, регистры палитры. С этого момента прикладная программа может использовать любую функцию, прототип которой есть в заголовочном файле <graphics.h>.

Если при выполнении инициализации возникает противоречие между запрашиваемым режимом и типом видеоадаптера, либо отсутствует достаточный объем свободной оперативной памяти и т.п., функция устанавливает код ошибки во внешней переменной, доступной после вызова функции graphresult(). Кроме того, код ошибки передается в точку вызова в ячейке памяти, на которую указывает graphdriver.

Если функции графической библиотеки больше не нужны прикладной программе, следует вызвать функцию closegraph() "закрытия" графического режима и возвращения к текстовому режиму. Эта функция освобождает память, распределенную под драйверы графики, файлы шрифтов и промежуточные данные и восстанавливает режим работы адаптера в то состояние, в котором он находился до выполнения инициализации системы.

Наиболее защищенный способ использования функции инициализации требует предварительного уточнения типа адаптера дисплея, активного в текущий момент времени. Для этого либо вызывается функция initgraph() со значением для graphdriver, равным указателю на DETECT, либо явно вызывается функция detectgraph(). Только после определения типа адаптера и его максимального режима выполняются установка нужного пользователю режима и загрузка .BGI-драйвера. Далее приводится описание функции detectgraph(). Она определяет тип активного видеоадаптера системы и тип подключенного монитора в персональном компьютере. Затем устанавливает тип подходящего для комбинации адаптер/монитор .BGI-драйвера и режим, обеспечивающий максимальное разрешение (максимальный режим). Например, если активным является CGA-адаптер, C++ считает режим 640 x 200 максимальным. Информация о подходящем драйвере и максимальном режиме возвращается в точку вызова в двух переменных, на которые указывают graphdriver и graphmode соответственно.

Прикладная программа может интерпретировать тип драйвера и максимальный режим, сравнивая возвращаемые значения с символическими константами, приведенными в табл. 3.1. и 3.2. В случае, если адаптер не способен работать ни в одном из графических режимов, функция устанавливает внутреннюю переменную кода ошибки в значение, равное grNotDetected (-2). На это же значение будет указывать и graphdriver при завершении функции.

Как отмечалось ранее, функция detectgraph() вызывается автоматически из функции инициализации видеосистемы initgraph(), если последняя вызывается со значением для graphdriver, равным указателю на DETECT. В

отличие от функции detectgraph() функция инициализации продолжает свою работу, загружает драйвер и устанавливает максимальный режим, рекомендованный (возвращенный) функцией detectgraph(). Функция detectgraph(), вызванная явно, не производит загрузку драйвера или установку режима. Для этого прикладная программа выполняет обращение к функции инициализации. В случае, если для функции initgraph(), вызываемой после явного обращения к detectgraph(), передаются параметры, возвращенные detectgraph(), получается такой же результат, что и при обращении к initgraph() с параметром graphdriver, равным указателю на DETECT. В этой связи раздельное обращение к detectgraph() и initgraph() имеет смысл лишь в случае, когда предполагается установка режима адаптера, отличающегося от максимального, т.е. если неприемлемы автоматические выбор и установка режима адаптера.

Защищенное от ошибок построение программы требует использования функции graphresult() после любого обращения к функциям detectgraph() и initgraph().

После того, как проведена инициализация графической системы, может быть установлен другой, не превосходящий максимального, режим видеоадаптера и выбраны цвета для пикселов. Установку режима выполняет функция setgraphmode(). 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.

После инициализации системы графики и установки нужного видеорежима возможен выбор необходимых цветов пикселов. Возможности по выбору цветов принципиально различны для CGA-, EGA- и VGA-адаптеров, что обусловлено различной логикой построения аппаратных средств.

Окно экрана в графическом режиме, или графическое окно (viewport), - это прямоугольная область экрана, заданная пиксельными координатами левого верхнего и правого нижнего углов. В графическом окне определены

относительные координаты. C++ позволяет выполнять вывод текста и графических примитивов в графическое окно. При этом по желанию пользователя вывод, не вмещающийся в границы окна, может усекаться.

Графическое окно может иметь отличающиеся от других участков экрана цвета фона и пикселов, маску заполнения и другие характеристики.

Для описания окна используется функция setviewport(). Текущие характеристики окна доступны программе через обращение к функции getviewsettings().

Левый верхний угол окна рассматривается как начало относительных координат X и Y всеми функциями графического вывода, в том числе и при выводе текста в графических режимах. Сразу после инициализации системы графики графическое окно охватывает весь экран, и, таким образом, началом графических координат по умолчанию является самый левый верхний угол экрана. Основное применение функции - определение и сохранение характеристик текущего графического окна перед переопределением текущего окна для последующего восстановления параметров окна.

Графические координаты X и Y измеряются в пикселах экрана относительно координат левого верхнего угла текущего окна. Функции графического вывода изменяют эти координаты в соответствии с объемом выведенной на экран информации. Текущие координаты в окне доступны через функции getx() и gety(). Установку нужных значений координат текущей позиции выполняют функции moveto() и moverel(). Кроме того, некоторые функции графического вывода позволяют задать текущую позицию (см., например, outtextxy()).

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 (рисунок 1).

6	$\cos^2(x/4)+\mathrm{Sqrt}(x)$	$3\pi/2$	16π

Рисунок 1 – Заданная функция (вариантное задание)

2. Произвести разметку осей и проставить истинные значения точек.

3. Найти максимальное значение функции на заданном интервале и вывести в отдельное окно на экране.

Структурная схема аппаратных средств

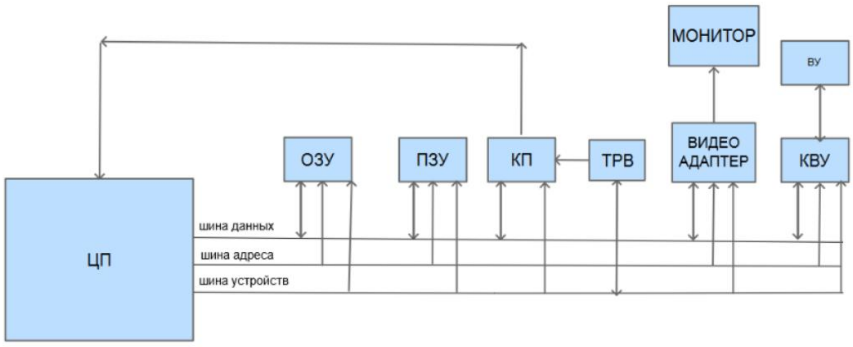

Рисунок 2 – Структурная схема аппаратных средств

Пример запуска программы

1. При запуске программы отображается график заданной функции согласно варианту (рисунок 3):

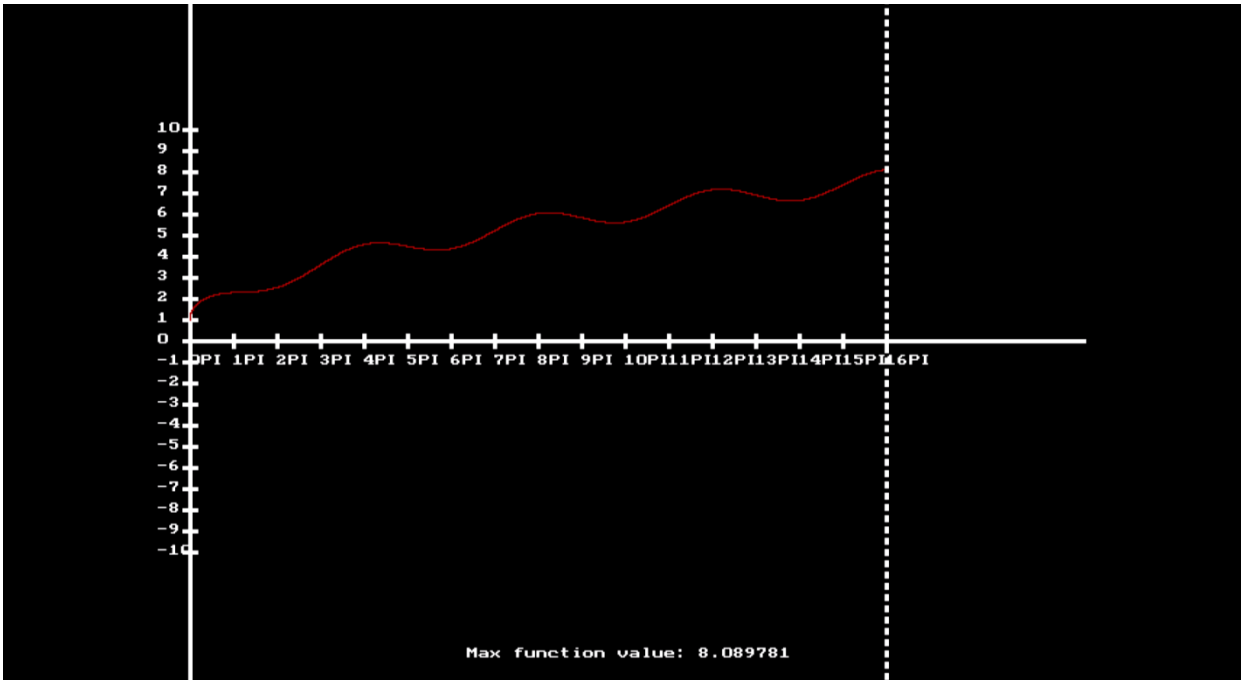

Рисунок 3 – Отображение графика функции

Вывод

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++.

ПРИЛОЖЕНИЕ А
РАБОЧИЙ КОД

```
#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 = 30 / PI;  // КОЭФФИЦИЕНТ ДЛЯ МАСШТАБИРОВАНИЯ ПО X
   CONST DOUBLE Y_APPROACH = 15;   // КОЭФФИЦИЕНТ ДЛЯ МАСШТАБИРОВАНИЯ ПО Y
   CONST INT TEN_NUM = 10;
   CONST FLOAT START_OUT = 0;       // НАЧАЛО ДИАПАЗОНА В ПИ
   CONST INT END_OUT = 16;          // КОНЕЦ ДИАПАЗОНА В ПИ (16*PI)
   CONST INT HATCH_OUT = 30;
   CONST INT NUMBER_OUT = 10;
   CONST INT LINES_OUT = 22;
   DOUBLE Y_RESULT;                 // Y КООРДИНАТА
   DOUBLE X_RESULT;                 // X КООРДИНАТА
   DOUBLE Y_OUT;                    // Y КООРДИНАТА ДЛЯ ВЫВОДА
   DOUBLE X_OUT;                    // X КООРДИНАТА ДЛЯ ВЫВОДА
   DOUBLE MAX_RESULT = -100;        // МАКСИМУМ ФУНКЦИИ

   INT I;

   INT GRAPH_DRIVER, GRAPH_MODE, GRAPH_ERROR_CODE;

   INT MAX_X;                       // МАКСИМАЛЬНАЯ X КООРДИНАТА ОКНА
   INT MAX_Y;                       // МАКСИМАЛЬНАЯ Y КООРДИНАТА ОКНА

   CHAR SYMBOLS_OUT[10];            // СТРОКА ДЛЯ ОТОБРАЖЕНИЯ ЗНАЧЕНИЙ ОСЕЙ
   CHAR MAXIMUM_OUT[50];            // СТРОКА ДЛЯ ОТОБРАЖЕНИЯ МАКСИМУМА

   CLRSCR();
   GRAPH_DRIVER = DETECT;
   DETECTGRAPH(&GRAPH_DRIVER, &GRAPH_MODE);
   INITGRAPH(&GRAPH_DRIVER, &GRAPH_MODE, "C:\\TURBOC3\\BGI");

   // ПРОВЕРКА НА ОШИБКУ ИНИЦИАЛИЗАЦИИ ГРАФИКИ
   GRAPH_ERROR_CODE = GRAPHRESULT();
   IF (GRAPH_ERROR_CODE != GROK) {
      PRINTF("ОШИБКА: %S\N", GRAPHERRORMSG(GRAPH_ERROR_CODE));
      GETCH();
      RETURN 255;
   }

   // ПОЛУЧАЕМ МАКСИМАЛЬНЫЕ ЗНАЧЕНИЯ КООРДИНАТ
   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);          // ВЕРТИКАЛЬНАЯ ОСЬ
   LINE(LINES_OUT, MAX_Y / 2, MAX_X, MAX_Y / 2);       // ГОРИЗОНТАЛЬНАЯ ОСЬ

   // РАЗМЕТКА ОСИ X (ОТ 0 ДО 16П)
   FOR (I = 0; I <= END_OUT; I++) {
```

```
            SPRINTF(SYMBOLS_OUT, "%DPI", I);
            OUTTEXTXY(LINES_OUT + HATCH_OUT * I, MAX_Y / 2 + TEN_NUM, 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);
        }

        // РИСУЕМ ПОСЛЕДНЮЮ ОТМЕТКУ ДЛЯ 16П, ЕСЛИ ОНА НЕ ПОПАЛА В ЦИКЛ
        IF (END_OUT == 16) {
            SPRINTF(SYMBOLS_OUT, "16PI");
            OUTTEXTXY(LINES_OUT + HATCH_OUT * 16, MAX_Y / 2 + TEN_NUM, SYMBOLS_OUT);
            LINE(LINES_OUT + HATCH_OUT * 16, MAX_Y / 2 + TEN_NUM / 2, LINES_OUT + HATCH_OUT * 16, MAX_Y / 2 - TEN_NUM /
2);
        }

        // РАЗМЕТКА ОСИ Y (ОТ -10 ДО 10)
        FOR (I = -NUMBER_OUT; 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);
        }

        // РИСУЕМ АСИМПТОТЫ
        SETLINESTYLE(3, 0, 3);
        LINE(LINES_OUT + HATCH_OUT * START_OUT, MAX_Y, LINES_OUT + HATCH_OUT * START_OUT, 0); // НАЧАЛЬНАЯ
        SETLINESTYLE(0, 0, 3);
        SETVIEWPORT(0, 0, MAX_X, MAX_Y, 0);

        // РИСУЕМ ГРАФИК ФУНКЦИИ
        FOR (X_RESULT = START_OUT * PI; X_RESULT <= END_OUT * PI; X_RESULT += 0.001) {
            Y_RESULT = POW(COS(X_RESULT / 4), 2) + SQRT(X_RESULT); // ФУНКЦИЯ
            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); // РИСУЕМ ПИКСЕЛЬ ГРАФИКА
        }

        // РИСУЕМ КОНЕЧНУЮ АСИМПТОТУ
        SETLINESTYLE(3, 0, 3);
        LINE(LINES_OUT + HATCH_OUT * END_OUT, MAX_Y, LINES_OUT + HATCH_OUT * END_OUT, 0);

        // ВЫВОД МАКСИМУМА ФУНКЦИИ
        SPRINTF(MAXIMUM_OUT, "MAX FUNCTION VALUE: %F", MAX_RESULT);
        OUTTEXTXY(MAX_X / 3, MAX_Y - LINES_OUT, MAXIMUM_OUT);

        GETCH();
        CLOSEGRAPH();
        RETURN 0;
}
```