МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный
электротехнический университет «ЛЭТИ»

Электронные методические указания к выполнению лабораторных работ по дисциплине «Организация ЭВМ и систем»

Санкт-Петербург
Издательство СПБГЭТУ «ЛЭТИ»
2013

УД

Лабораторная работа № 3.

ИССЛЕДОВАНИЕ ВИДЕОСИСТЕМЫ (ГРАФИЧЕСКИЙ РЕЖИМ)

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

3.1. Общие положения

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++. Подробно эти элементы модели рассматриваются в следующих подразделах.

3.2. Инициализация и закрытие системы графики

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.

Табл. 3.1. Видеорежимы в библиотеке графики

Константа режима	Характеристика режима	Номер режима	Тип адаптера и драйвер
CGAC0 CGAC1 CGAC2 CGAC3	320x200, палитра 0 320x200, палитра 1 320x200, палитра 2 320x200, палитра 3	4,5	CGA, EGA, VGA, MCGA и др. в режиме эмуляции CGA. Используется CGA.BGI
CGAHI	640x200, 2 цвета	6	
MCGAC0 MCGAC1 MCGAC2 MCGAC3	320x200, палитра 0 320x200, палитра 1 320x200, палитра 2 320x200, палитра 3	4,5	MCGA. Используется MCGA.BGI
MCGAMED	640x200, 2 цвета	6	
MCGAHI	640x480, 2 цвета	11h	
EGALO	640x200, 16 цветов	0Eh	EGA с памятью >128K байт, VGA при эмуляции EGA. Используется EGAVGA.BGI
EGAHI	640x350, 16 цветов	10h	
EGA64LO	640x200, 16 цветов	0Eh	EGA с памятью 64K байт, VGA при эмуляции EGA. Используется EGAVGA.BGI
EGA64HI	640x350, 4 цвета	10h	
EGAMONOHI	640x350, 2 цвета	0Fh	EGA, VGA при эмуляции EGA Используется EGAVGA.BGI
HERCMONOHI	720x348.	7h	
ATT400C0 ATT400C1 ATT400C2	320x200, палитра 0 320x200, палитра 1 320x200, палитра 2	4,5	AT&T. Используется ATT.BGI
VGALO	640x200, 16 цветов	0Eh	VGA. Используется EGAVGA.BGI
VGAMED	640x350, 16 цветов	10h	
VGAHI	640x480, 16 цветов	12h	
PC3270HI	720x350, 1 с.	?	IBM PC 3270. Используется PC3270.BGI
IBM8514LO	640x480, 256 цветов	?	IBM 8514. Используется IBM8514.BGI
IBM8514HI	1024x768, 256 цветов	?	IBM 8514. Используется IBM8514.BGI

Примечание. Символом “?” обозначены режимы, не вошедшие в табл. 2. 1 .

Инициализацию графической модели выполняет функция initgraph().

void far initgraph(int *graphdriver, int *graphmode, char * pathtodriver).

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.

Табл. 3.2. Задание используемого .BGI-драйвера

Символическая константа из graphics_drivers	Значение (в 10 с/с)	Описание
DETECT	0	Запрос автоматического определения типа драйвера
CGA	1	Загрузка драйвера для CGA-адаптера или переключение старших адаптеров в режим эмуляции CGA и загрузка драйвера для CGA-адаптера
MCGA EGA	2 3	Загрузка драйвера для MCGA-адаптера Загрузка драйвера для EGA-адаптера с объемом видеопамяти 128К байт и более и ECD-монитором
EGA64	4	Загрузка драйвера для EGA-адаптера с объемом видеопамяти 64К байт и ECD-монитором

Символическая константа из graphics_drivers	Значение (в 10 с/с)	Описание
EGAMONO	5	Загрузка драйвера для EGA-адаптера с объемом видеопамяти 64К байт и монохроматическим монитором
IBM8514	6	Загрузка драйвера для адаптера IBM 8514 с аналоговым монитором
HERCMONO	7	Загрузка драйвера для адаптера HGC с монохроматическим монитором
ATT400	8	Загрузка драйвера для графического адаптера AT&T с разрешением 400 линий
VGA	9	Загрузка драйвера для VGA-адаптера с аналоговым монитором
PC3270	10	Загрузка драйвера для графического адаптера IBM PC 3270 с аналоговым монитором

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.

closegraph().

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.

Приведем "скелет" программы, выполняющей все необходимые подготовительные действия для использования функций библиотеки графики. Для опреде-

ления типа видеоадаптера в ней используется функция initgraph() .

```
#include <graphics.h> /* все графические функции используют данный заголовочный файл */
int main(void)
{
int graph_driver;          /* используемый драйвер */
int  graph_mode;                    /* графический режим видеоадаптера */
int graph_error_code;  /* внутренний код ошибки */
/* Определение типа видеоадаптера, загрузка подходящего .BGI-драйвера и установка максимального режима. Считается, что драйвер находится на диске d: в директории \bc\bgi. */ graph_driver = DETECT;
initgraph(&graph_driver, &graph_mode, "d:\\bc\\bgi" );
/* Определение кода ошибки при выполнении инициализации. */
graph_error_code = graphresult( );
      if(graph_error_code != grOk)       /* всегда следует проверять наличие ошибки !                */
{
/* Обработка ошибки . return 255; */
return 255;
      }
```

/* Установка в случае необходимости режима, отличающегося от максимального; выбор палитры, цвета, стиля линий, маски заполнения и других параметров, отличающихся от значений по умолчанию. Вывод графических примитивов: прямых линий, окружностей, эллипсов, столбцовых диаграмм и т.п. */

```
/* Закрытие графического режима */
closegraph();
}
```

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().

```
void detectgraph (int *graphdriver, int *graphmode).
```

Определяет тип активного видеоадаптера системы и тип подключенного мо-

нитора 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.

3.3. Обработка ошибок системы графики

Защищенное от ошибок построение программы требует использования функции graphresult() после любого обращения к функциям detectgraph() и initgraph(). Далее следует описание функций обработки ошибок, сообщающих внутренние коды ошибок графической библиотеки (graphresult()) или формирующей строку диагностического сообщения (grapherrormsg()).

```
int graphresult(void)
```

Возвращает значение внутреннего кода ошибки, установленного последним обращением к функциям графической библиотеки. Перед завершением сбрасывает код ошибки в 0. Прикладная программа может интерпретировать возвращаемое значение, сопоставляя его с целым числом либо с символической константой из перечислимого типа graphics_errors, определенного в <graphics.h> (табл. 3.3).

char * grapherrormsg(int errorcode)

Возвращает указатель на ASCII-строку символов, содержащую сообщение об ошибке, соответствующее внутреннему коду ошибки errorcode функций графики Turbo C. Функция grapherrormsg() возвращает указатели на сообщения на английском языке. В принципе несложно выполнить их "перевод" непосредственно, переработав саму функцию grapherrormsg().

Табл. 3.3. Коды ошибок, возвращаемые при выполнении функций графической библиотеки.

Символическая константа из graphics_errors	Значение (в 10 с/с)	Описание
grOk	0	Отсутствие ошибки
grNoInitGraph	-1	Графический интерфейс (.BGI-драйвер) не инсталлирован. Следует выполнить initgraph()
grNotDetected	-2	Не обнаружен видеоадаптер, способный работать в запрошенном (или любом в случае DETECT) графическом режиме
grFileNotFound	-3	Не найден по заданному маршруту и в текущем директории .BGI-файл
grlnvalidDriver	-4	Заданный в качестве .BGI-драйвера файл не соответствует стандарту Turbo C
grNoLoadMem	-5	Недостаточно свободной памяти для загрузки драйвера и хранения промежуточных результатов
grNoScanMem	-6	Нехватка памяти при выполнении графических функций заполнения
grNoFloodMem	-7	Нехватка памяти при выполнении графических функций заполнения
grFontNotFound	-8	Не найден файл описания шрифта
grNoFontMem	-9	Отсутствие памяти для загрузки файла шрифта
grlnvalidMode	-10	Недопустимый графический режим для выбранного .BGI-драйвера
grError	-11	Ошибка функции графики
grIOerror	-12	Ошибка ввода-вывода в графическом режиме

Символическая константа из graphics_errors	Значение (в 10 с/с)	Описание
grInvalidFont	-13	Файл шрифта, не соответствующий стандарту Borland International
grInvalidFontNum	-14	Недопустимый номер шрифта
grInvalidDeviceNum	-15	Недопустимый номер устройства
grInvalidVersion	-18	Недопустимый номер версии .BGI-драйвера (.BGI-драйвер для версии 1.5)

3.4. Определение и установка графического режима

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. Далее следует описание упомянутых функций.

int getgraphmode (void)

Возвращает текущий графический режим, установленный для графической модели функциями initgraph() или setgraphmode(). Возвращаемое значение соответствует номеру режима, установленному для инсталлированного драйвера графики. Возвращаемое значение соответствует числовому значению символических констант режима, перечисленных в табл. 3 .1.

int getmaxmode(void)

Возвращает число, определяющее максимально возможный для инсталлированного .BGI-драйвера режим. Как и в предыдущем случае, возвращаемое значение соответствует номеру режима, установленному для инсталлированного драйвера графики. Возвращаемое значение соответствует числовому значению символических констант режима, перечисленных в табл. 3.1.

int getmaxx(void)

int getmaxy(void)

Возвращают максимальные значения координат X и Y для текущего видеорежима. Например, для режима CGA0 getmaxx() возвращает значение 319, а getmaxy() -199. Функции особенно полезны для центрирования изображений и определения таких размеров знакомест при выводе текста в графическом режиме, чтобы текст помещался в заданную область экрана.

```
char * getmodename(int mode_number)
```

Возвращает указатель на ASCII-строку символов, содержащую имя символической константы, соответствующей режиму mode_number. Значение mode_number должно быть в пределах диапазона значений, возвращенных функцией getmaxmode() (для любого драйвера) или getmoderange() (для драйвера Borland International).

```
void setgraphmode(int mode)
```

Устанавливает видеосистему в режим, заданный значением переменной mode, и сбрасывает значения внутренних переменных системы графики в их значения по умолчанию (стиль линий, маска заполнения, шрифт и т.д.). Значение mode соответствует числовому значению символических констант режима, перечисленных в табл. 3.1. При задании недопустимого режима для текущего .BGI-драйвера функция устанавливает внутренний код ошибки -1 (см. табл. 3.3).

Обычно функция используется для обратного переключения в графический режим после того, как видеоадаптер был на время переключен функцией restorecrtmode() в текстовый режим. Однако функция может использоваться для переключения и из одного графического режима в другой, не выходящий за диапазон допустимых режимов.

```
void restorecrtmode(void)
```

Возвращает видеоадаптер в режим, в котором он был до выполнения инициализации системы графики. Как правило, исходным режимом будет текстовый. В том случае, если выполняется временный возврат в исходный (текстовый) режим, перед выполнением функции restorecrtmode() в переменной следует сохранить текущий графический режим, используя, например, обращение к функции getgraphmode(), после чего возможен возврат в графический режим с помощью функции setgraphmode().

3.5. Управление цветами и палитрами

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.

Далее приведена спецификация функций библиотеки графики для работы с цветами и палитрами.

int getbkcolor(void)

Возвращает целое число, равное коду цвета фона.

int getmaxcolor(void)

Возвращает максимальное значение кода цвета пиксела минус 1. Это значение позволяет установить максимальное число цветов, которое может отображаться на экране. В зависимости от режима, в котором проведена инициализация системы графики, возвращаемое значение может быть равно 1, 3 или 15.

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. До того момента, пока цвет не установлен, используется максимальный (из палитры) номер цвета. В случае, если color задает недопустимый номер цвета для текущей палитры, текущий цвет остается неизменным.

3.6. Задание окна экрана. Определение и установка графических координат

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 харак-

теристики.

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. Описание структурной переменной выполняет вызывающая сторона. Функции передается указатель на описанную переменную. Шаблон viewporttype описан в <graphics.h>:

```
struct viewporttype
{
int left, top;        /* координаты ( столбец, строка) левого верхнего угла* /
int right, bottom; /* координаты (столбец, строка) правого нижнего угла */
int clip;             /* Флаг усечения при выводе (1 - усечение, 0 - нет )*/
 }
```

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.

```
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. В качестве начала текущих координат для функций графического вывода устанавливается левый верхний угол.

Помимо явного задания окна функцией setviewport(), оно специфицируется и неявно при выполнении функций initgraph(), setgraphmode() и graphdefaults(). При каждом их выполнении в качестве графического окна устанавливается весь экран.

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()).

```
int getx (void)
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. Аргументы x, y задают новые значения координат текущей позиции относительно координат левого верхнего угла текущего графического окна.

```
void moverel(int dx, int dy)
```

Устанавливает новое значение координат текущей позиции. Аргументы dx, dy задают новые значения координат относительно текущих координат графического окна. Другими словами, новая текущая позиция устанавливается в точку, отстоящую от текущей позиции на dx столбцов пикселов по горизонтали и dy строк пикселов по вертикали. Чтобы переместить текущую позицию влево, нужно задать для dx отрицательное значение. Для перемещения текущей позиции по вертикали вверх задается отрицательное значение для dy.

3.7. Вывод текста в графическом режиме видеоадаптера

C++ предоставляет пользователю широкие возможности по выводу текстовой информации в графических режимах. Во-первых, это все функции стандартного вывода.

Библиотека графики позволяет выводить на экран текст различными шрифтами. C++ имеет два типа шрифтов: битовый и сегментированный.

Каждому символу битового шрифта (bit-mapped font) ставится в соответствие матрица пикселов фиксированного размера. C++ использует в качестве битового шрифта таблицу знакогенератора для символов размером 8x8, установленную в компьютере перед инициализацией системы графики. Все изменения таблицы знакогенератора, сделанные, например, программами русификации, будут сохранены. Это позволяет, применяя функции, выводить текст русскими буквами и в

графических режимах.

Другой тип шрифтов, используемый при выводе текста на экран, фактически задает правило "рисования" каждого символа. Он описывается как совокупность отрезков прямых линий, или сегментов. Этим и объясняется название шрифта - сегментированный (stroke font). 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 C++ доступны четыре сегментированных шрифта: Triplex, Small, Sans-Serif и Gothic. Файлы сегментированных шрифтов располагаются в файлах с расширением .CHR и, подобно .BGI-драйверам, загружаются как оверлеи во время исполнения программы. Библиотека графики дает возможность выводить текст в графических режимах слева направо и снизу вверх. Для вывода символов при использовании любого шрифта может быть задан масштаб знакоместа по отношению к знакоместу шрифта 8x8. При использовании сегментированного шрифта может быть задан размер знакоместа и в относительных единицах масштаба как по вертикали, так и по горизонтали. Кроме того, выводимые строки текста могут выравниваться по-разному: строка может быть "прижата" влево и вверх относительно точки, определенной как текущая точка отсчета, влево и вниз и т.п.

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. Функции передается указатель texttypeinfo на описанную структурную переменную. Шаблон textsettingstype описывается в <graphlcs.h> так:

```
struct textsettingstype
{
int font;         /* номер шрифта из перечислимого типа font_names[] */
int direction;  /* направление вывода текста (гор. или верт.) */
 int charsize;  /* размер знакоместа относительно шрифта 8x8 */
```

```
  int horiz;       /* код выравнивания по горизонтали (влево, вправо) */
  int vert;        /* код выравнивания по вертикали (вверх, вниз) */
 }
```

Интерпретация отдельных полей структурной переменной по шаблону textsettingstype приведена в описании функции settextstyle().

```
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 C. Возможные типы шрифтов задаются либо целым числом, либо символической константой из перечислимого типа font_names. В табл. 3.4. приведены доступные шрифты, соответствующие им символические константы, числовые значения и имена файлов шрифтов.

Табл. 3.4. Шрифты, доступные в C++

Символическая константа из font_names	Значение	Описание шрифта	Имя файла шрифта
DEFAULT FONT	0	Битовый шрифт 8x8	-
TRIPLEX_FONT	1	Сегментированный шрифт Triplex	TRIP.CHR
SMALL_FONT	2	Сегментированный шрифт Small	SMAL.CHR
SANS_SERIF_FONT	3	Сегментированный шрифт Sans-Serif	SANS.CHR
T GOTHIC_FONT	4	Сегментированный шрифт Gothic	GOTH.CHR

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.

Третий аргумент функции - charsize - задает масштаб каждого символа относительно знакоместа 8x8. Если, например, задать charsize равным 5, то символ будет изображаться в знакоместе 40 x 40. Если charsize равен 0, то для битового шрифта размер знакоместа изменяться не будет. Для сегментированного шрифта размер знакоместа будет определяться значениями, установленными функцией setusercharsize().

В случае ошибки функция settextstyle() устанавливает во внутренней переменной системы графики соответствующий код ошибки. Он доступен программе

через обращение к функции graphresult().

Установленные в системе графики значения высоты символа и его ширины можно получить через обращения к функциям textheight() и textwidth().

int textheight(char *textstring)

Возвращает высоту строки символов в пикселах, на которую указывает textstring. Использует информацию о текущем шрифте и установках масштаба знакоместа. Сама строка, на которую указывает textstring, на экран не выводится. Наиболее часто функция используется для установки нужных промежутков между строками текста, а также при вычислении таких масштабов для символов, которые позволяли бы уместить нужное число строк в фиксированной области экрана.

int textwidth (char far *textstring)

Возвращает ширину строки символов в пикселах, на которую указывает textstring. Использует информацию о текущем шрифте и установках масштаба знакоместа. Сама строка, на которую указывает textstring, на экран не выводится. Наиболее часто функция используется для установки нужных промежутков между символами текста, а также при вычислении таких масштабов для символов, которые позволяли бы уместить нужное число строк в фиксированной области экрана. В частности, обращение к функции

textwidth("A");

возвращает ширину символа 'A' в пикселах.

Еще одна установка системы графики, затрагивающая вывод текста в текстовом режиме, это выравнивание символов. Специальная функция библиотеки графики settextjustify() позволяет изменить установку по умолчанию для выравнивания символов при выводе текста в графических режимах.

void far settextjustify(int horiz,int vert)

Задает новую установку выравнивания символов текста в графических режимах работы адаптера. Она выполняет выравнивание всей строки символов, выводимой функциями outtext() или outtextxy() относительно некоторой точки, называемой далее точкой отсчета. Координаты точки отсчета задаются либо явно

функцией outtextxy(), либо используются текущие значения.

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_TEXT - правая граница строки "прижимается" слева к вертикальной линии, проведенной через точку отсчета.

Аргумент vert также может принимать три значения, задаваемых символическими константами: BOTTOM_TEXT - 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; TOP_TEXT - верхняя граница строки "прижимается" снизу к горизонтальной линии, проведенной через точку отсчета.

Отметим, что при выводе текста вертикально установки выравнивания по горизонтали LEFT_TEXT и RIGHT_TEXT не различаются библиотекой графики Turbo C и аналогичны RIGHT_TEXT.

После того, как заданы все необходимые параметры текста, можно выводить ASCII-строки, используя функцию outtext() или outtextxy().

```
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. В случае, когда текст выводится горизонтально и установлено выравнивание LEFT_TEXT, функция outtext() продвигает координату X текущей позиции на значение, равное textwidth(textstring). В остальных случаях координата X текущей позиции остается неизменной. Если текст выводится в графическое окно с включенным усечением, текст усекается на границах окна. Для сегментированных шрифтов усечение производится с точностью до пикселов, для битовых шрифтов оно происходит с точностью до символа. Усечение строки может выполняться по одной границе или по обеим границам сразу в случае, когда задается выравнивание по центру.

```
void outtextxy (int x, int y, char *textstring)
```

Выводит ASCII-строку текста, на начало которой указывает textstring, ис-

пользуя текущие цвет, установки направления, типа шрифта и выравнивания строки. Аргументы x и y явно специфицируют новую текущую позицию, используемую для вывода строки. Координаты X и Y измеряются относительно координат левого верхнего угла текущего графического окна. В случае, когда текст выводится горизонтально и установлено выравнивание LEFT_TEXT, функция outtext() продвигает координату X текущей позиции на значение, равное textwidth(textstring). В остальных случаях координата X текущей позиции остается неизменной. Если текст выводится в графическое окно с включенным усечением, он усекается на границах окна. Для сегментированных шрифтов усечение производится с точностью до пикселов, для битовых шрифтов оно происходит с точностью до символа. В случае, когда установлено выравнивание CENTER_TEXT, но выводимая строка не помещается в текущем графическом окне, функция не выполняет вывод.

Функции способны выводить только нуль-терминированные строки, и для выполнения форматированного вывода в графических режимах выбранными сегментированными шрифтами поступают следующим образом. Сначала, используя функцию стандартного вывода sprintf(), получают нужную форматную строку, а затем выводят ее с помощью функции outtextxy() выбранным шрифтом.

3.8. Вывод графической информации

3.8.1. Параметры и атрибуты графического вывода

Если инициализация системы графики выполнена успешно, становятся доступными функции графической библиотеки для построения основных графических примитивов - отрезков прямых линий, дуг, окружностей, эллипсов, прямоугольников, секторных и столбцовых диаграмм и т.д. Многие из этих фигур могут быть по желанию программиста "залиты" текущим цветом с использованием текущего шаблона или маски заполнения. Специальные функции позволяют запоминать прямоугольные области экрана и затем восстанавливать их в специфицированном месте экрана.

Все функции библиотеки графики, генерирующие вывод информации на экран, работают в пределах текущего графического окна. Для графического вывода используется текущий цвет пиксела, установленный функцией setcolor().

При выводе отрезков прямых линий и графических примитивов система графики позволяет определить такой параметр, как стиль линии. C++ поддерживает ряд предопределенных стилей линий. Как и в случае маски заполнения, пользователь может описать собственный стиль линии. Для определения текущей установ-

ки 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. Функция заполняет поля структурной переменной по шаблону struct linesettingstype. Структурную переменную описывает точка вызова и передает в функцию указатель lineinfo на эту переменную. Шаблон struct linesettingstype описан в заголовочном файле <graphics.h> следующим образом:

```
struct linesettingstype
{
                    int linestyle;     /*идентифицирует стиль линии*/
 unsigned upattern;    /*задает определенный пользователем стиль линии.
Имеет значение      только тогда, когда linestyle = USERBIT_LINE*/
                    int thickness;    /*задает толщину линии*/
}
```

Стиль линии задается либо целым числом, либо символической константой по табл. 3.5.

Табл. 3.5. Задание стиля линий

Символическая константа	Значение	Описание стиля линии
SOLID_LINE	0	Сплошная (непрерывная) линия
DOTTED_LINE	1	Линия из точек ....
CENTER_LINE	2	Штрих-пунктирная линия -.-.
DASHED_LINE	3	Штриховая линия
USERBIT_LINE	4	Определенная пользователем линия. Ее шаблон описывает поле upattern в структуре linesettingstype

Толщина линий может быть равна 1 или 3 пикселам. Она задается либо целым числом, либо символической константой из табл. 3.6.

Табл. 3.6. Задание толщины линий в Turbo C

Символическая константа	Значение	Толщина
NORM_WIDTH	1	Задает толщину линии 1 пиксел
THICK_WIDTH	3	Задает толщину линии 3 пиксела

После инициализации системы графики для стиля линии устанавливается непрерывная линия толщиной 1 пиксел. Задание стиля линии выполняет функция setlinestyle().

```
void setlinestyle (int linestyle, unsigned upattern, int thickness)
```

Устанавливает стиль "рисования" отрезков прямых линий и графических примитивов. Аргумент linestyle выбирает стиль линии в соответствии с табл. 3.5., а аргумент thickness - толщину линии по табл. 3.6.

Аргумент upattern используется только в том случае, когда задается отличный от предопределенных стиль линии, т.е. если linestyle равен USERBIT_LINE (4). При этом 16 бит аргумента upattern задают маску линии (светимость 16 подряд расположенных пикселов линии). Если бит в upattern равен 1, пиксел выводится на экран текущим цветом, установленным функцией setcolor(). Определенный пользователем стиль линии действует только в случае, когда устанавливается толщина линии 1 пиксел (NORM_WIDTH). Для толщины 3 пиксела определенный пользователем стиль линии не действует.

При выводе отрезков прямых линий в графическом режиме система графики позволяет задать дополнительно режим вывода линии. Существуют два различных режима, устанавливаемых функцией setwritemode().

```
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.

Следующий параметр системы графики - так называемый коэффициент сжатия, или коэффициент пропорциональности (aspect ratio). Он задает форму пиксела на экране монитора. Для многих мониторов световое пятно, которое соответствует пикселу, не является строго квадратным, а напоминает по форме эллипс, вытянутый вверх. Как следствие этого, линия, состоящая из одного и того же чис-

ла пикселов, расположенная вертикально, выглядит на экране длиннее, чем линия из того же числа пикселов, расположенная горизонтально. По этой же причине вывод прямоугольника, имеющего равные (в пикселах) горизонтальную и вертикальную стороны, не приводит к получению на экране квадрата. Система графики учитывает коэффициент сжатия при выводе сложных графических примитивов - эллипсов, окружностей, дуг, круговых секторов. Благодаря этому они появляются на экране геометрически корректными. По умолчанию после инициализации системы графики автоматически устанавливает коэффициент сжатия в соответствии с характеристиками аппаратуры видеосистемы. Для управления коэффициентом сжатия в предусмотрены две функции: getaspectratio() и setaspectratio().

void getaspectratio (int *xasp, int *yasp)

Заполняет две переменные, описанные точкой вызова, значениями коэффициента сжатия для текущего видеорежима. Возвращаемые значения задают фактически физическую форму пиксела. Для размера пиксела по вертикали (значение, на которое указывает yasp), всегда возвращается 10 000. Если световое пятно на экране, соответствующее пикселу, является квадратным (как для адаптера VGA), то и значение "ширины" пиксела равно 10 000. Для других видеоадаптеров пиксел на экране имеет эллипсообразную форму с большой полуосью, сориентированной по вертикали. Для таких адаптеров в ячейке, на которую указывает xasp, возвращается значение, меньшее 10 000. Зная значения геометрических размеров пиксела, можно так скорректировать параметры для функций вывода, чтобы получить на экране геометрически пропорциональные фигуры, например квадраты. Отметим, что не следует корректировать коэффициент сжатия для вывода окружностей, дуг или секторных диаграмм. Система графики делает корректировку автоматически.

void setaspectratio (int xasp, int yasp)

Устанавливает новое значение коэффициента сжатия, которое будет использоваться системой графики при выводе геометрических примитивов - прямоугольников, дуг, окружностей, эллипсов. Аргумент xasp отображает в условных единицах ширину пиксела на экране, yasp - высоту пиксела. Например, если известно, что высота пиксела на экране в 1.2 раза больше, чем его ширина, геометрически корректный вывод будет получен при задании такого коэффициента сжатия:

setaspectratio(100,120);

Рекомендуемое использование функции - 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.

Последние из параметров графической системы, влияющие на вывод графической информации, это маска заполнения и стиль заполнения. Маска заполнения позволяет задать способ заполнения отдельных областей экрана. Она определяется восьмибайтовым шаблоном, рассматриваемым как битовая карта 8x8. Заполняемая область также разбивается на блоки (знакоместа) по 8x8 пикселов. Маска "накладывается" на каждое такое знакоместо по следующему правилу: если соответствующий бит в маске заполнения равен 1, то пиксел в знакоместе имеет код текущего цвета; в противном случае пиксел остается неизменным. Для работы с масками заполнения система графики содержит функции getfillpattern() и setfillpattern().

```
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. Маска заполнения может иметь одно из предопределенных значений или описываться пользователем.

```
void setfillpattern (char *upattern, int color)
```

Задает цвет пикселов и маску для заполнения областей экрана. По умолчанию используется белый цвет и маска заполнения, состоящая из матрицы единиц во всех битах. Таким образом, по умолчанию все пикселы заполняемой области имеют белый цвет. Аргумент upattern указывает на начало области из 8 байт, задающих новую маску заполнения. Первый байт задает пикселы самой верхней строки в пределах знакоместа. Старший бит первого байта соответствует самому левому пикселу знакоместа. Аргумент color задает цвет пикселов.

Для удобства пользователей библиотека графики содержит целую группу предопределенных комбинаций символ/цвет заполнения областей экрана. Пару значений символов/цветов часто называют стилем заполнения (filling style). Для работы с предопределенными стилями используется пара функций getfillsettings()

и setfillstyle().

```
void getfillsettings( struct fillsettingstype *fillinfo)
```

Заполняет поля структурной переменной по шаблону struct fillsettingstype информацией о текущей маске и цвете заполнения. Структурную переменную по шаблону struct fillsettingstype описывает точка вызова. Аргумент fillinfo указывает на описанную точкой вызова структурную переменную. Шаблон struct fillsettingstype определен в <graphics.h> так:

```
struct fillsettingstype
{
int pattern; /* идентификатор маски заполнения */
   int color;    /* цвет заполнения       */
}
```

Идентификатором предопределенных масок заполнения служит или целое число или символическая константа (табл. 3.7.).

В случае, когда используется определенная пользователем маска заполнения, поле pattern в структурной переменной, заполняемой функцией getfillsettings(), равно 12.

Табл. 3.7. Задание предопределенных масок

Символическая константа	Значение	Описание стиля заполнения
EMPTY FILL	0	Заполнение цветом фона
SOLID FILL	1	Заполнение текущим цветом
LINE FILL	2	Заполнение символами --, цвет - color
LTSLASH_FILL	3	Заполнение символами // нормальной толщины, цвет - color
SLASH_FILL	4	Заполнение символами // удвоенной толщины, цвет - color
BKSLASH_FILL	5	Заполнение символами \\ удвоенной толщины, цвет - color
LTBKSLASH_FILL	6	Заполнение символами \\ нормальной толщины, цвет - color
HATCH_FILL	7	Заполнение вертикально-горизонтальной штриховкой тонкими линиями, цвет-color
XHATCH_FILL	8	Заполнение штриховкой крест-накрест по диагонали "редкими" тонкими линиями, цвет - color
INTERLEAVE_FILL	9	Заполнение штриховкой крест-накрест по диагонали "частыми" тонкими линиями, цвет - color
WIDE DOT FILL	10	Заполнение "редкими" точками

Символическая константа	Значение	Описание стиля заполнения
CLOSE DOT FILL	11	Заполнение "частыми" точками
USER FILL	12	Заполнение по определенной пользователем маске заполнения, цвет - color

```
void setfillstyle(int pattern, int color)
```

Выбирает один из предопределенных стилей заполнения. Значение pattern идентифицирует стиль. Возможные значения для pattern приведены в табл. 3.7. Аргумент color задает цвет, используемый для пикселов по заданному шаблону. Данная функция не предназначена для установки определенной пользователем маски заполнения. Для этого используется функция setfillpattern().

3.8.2. Чтение-запись отдельных пикселов

Базовой функцией любой графической библиотеки является функция вывода в заданные координаты пиксела специфицированного цвета. C++ имеет в своем составе две функции манипуляции отдельными пикселами экрана: getpixel() - для определения кода цвета пиксела и putpixel () - для вывода пиксела текущим цветом.

```
unsigned getpixel( int x, int y)
```

Определяет, лежит ли пиксел с координатами (x, y) в текущем графическом окне, и, если лежит, возвращает код цвета этого пиксела. В противном случае возвращается 0.

```
void putpixel(int x, int y, int pixelcolor)
```

Определяет, лежит ли пиксел с координатами (x, y) в текущем графическом окне, и, если лежит, выводит на экран пиксел, код цвета которого равен pixelcolor. В противном случае цвет пиксела не изменяется.

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

Типичным применением точечного вывода является формирование сложных изображений, которые не могут быть представлены совокупностью графических примитивов: пиктограмм, фрагментов игрового поля и др. К сожалению, при ра-

боте с функцией putpixel() нельзя управлять режимом записи пиксела, т.е. функция putpixel() переопределяет предыдущее содержимое экрана. Это не всегда удобно, особенно в тех случаях, когда требуется добиться видимости изображения на любом фоне, другими словами, выполнить вывод пиксела, используя операцию исключающего ИЛИ с предыдущим содержимым.

3.8.3. Вывод отрезков прямых линий

Целая группа функций библиотеки графики предназначена для вывода отрезков прямых линий. Далее приводится спецификация этих функций. Напомним, что на вывод отрезков прямых линий влияют режим вывода линии и стиль линии.

Выводимые отрезки прямых линий не пересекают границ текущего окна, если при описании окна включен режим "усечения" (clipping).

```
void line( int x1, int y1, int x2, int y2)
```

Выводит отрезок прямой линии между двумя явно специфицированными точками (x1, y1) и (x2, y2), используя текущие цвет, стиль, толщину и режим вывода линии. Координаты (x1, y1) и (x2, y2) задаются относительно левого верхнего угла текущего графического окна. Функция не изменяет текущую позицию.

```
void linerel(int dx, int dy)
```

Выводит отрезок прямой линии между текущей позицией (начало отрезка) и точкой, заданной горизонтальным смещением dx и вертикальным смещением dy от текущей позиции (конец отрезка). При выводе отрезка прямой используются текущие цвет, стиль, толщина и режим вывода линии. После вывода линии функция устанавливает новую текущую позицию, равную координатам конца отрезка.

Установка необходимой текущей позиции может быть выполнена функциями moveto() и movrel().

```
void lineto( int x, int y)
```

Выводит отрезок прямой линии между текущей позицией (начало отрезка) и точкой, заданной горизонтальной координатой x и вертикальной координатой y (конец отрезка). При выводе отрезка прямой используются текущие цвет, стиль, толщина и режим вывода линии. Координаты (x, y) задаются относительно левого верхнего угла текущего графического окна. После вывода линии функция уста-

навливает новую текущую позицию, равную координатам конца отрезка.

3.8.4. Вывод основных графических примитивов

Библиотека графики содержит функции для вывода дуги окружности или целой окружности, эллиптической дуги или целого эллипса, кругового сектора, ломаной линии из нескольких отрезков прямой (полигона), прямоугольника, прямоугольной полосы заданного цвета и стиля заполнения, прямоугольника заданной толщины в аксонометрии.

Все примитивы, за исключением полосы и дуги, могут быть выведены контуром или их внутреннее пространство может быть "залито" заданным цветом и заполнено по текущей маске. Для изображения используется линия текущего стиля и толщины, для заполнения - текущие установки стиля заполнения (цвет, маска). Далее приводится описание функций для вывода примитивов.

```
void arc(int x, int y, int stangle, int endangle, int radius)
```

Выводит дугу окружности радиусом radius. Центр окружности задают координаты x, y. Аргументы stangle и endangle задают соответственно начальный и конечный углы (рис. 3.1.) выводимой дуги. Углы задаются в градусах и отсчитываются против хода часовой стрелки. Положению часовой стрелки 3 часа соответствует угол 0 градусов, 12 часов - 90 градусов, 9 часов - 180 градусов, 6 часов - 270 градусов. При задании stangle равным 0 градусов и endangle равным 359 градусов выводится полная окружность. Для вывода дуги используется текущий цвет и только сплошная линия. Толщина линии может быть задана функцией settextstyle() (1 или 3 пиксела). Текущая позиция при выводе дуги не изменяется. Функция автоматически корректирует координаты точек в соответствии с коэффициентом сжатия дисплея.

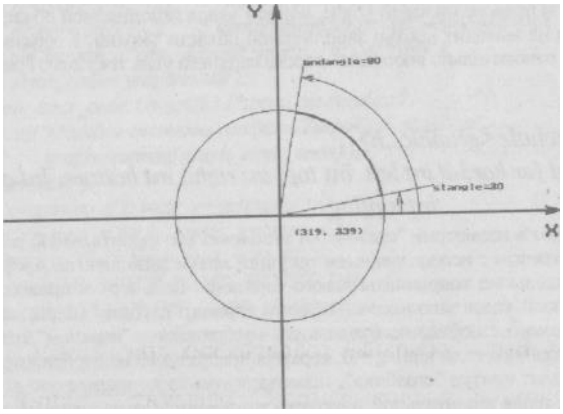

Рис. 3.1. Вывод окружностей и дуг

void bar(int left, int top, int right, int bottom)

Выводит полосу, заполненную текущим цветом с использованием текущей маски заполнения. Аргументы задают пиксельные координаты левого верхнего (left, top) и правого нижнего (right, bottom) углов заполняемой области экрана. Функция не выводит контур заполняемой области экрана. Координаты углов задаются относительно координат левого верхнего угла текущего графического окна.

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; необходимость изображения "верхней" поверхности столбца (topflag): если topflag = 0, верхняя поверхность не отображается. Функция выводит контур "столбика", используя только непрерывную линию. Координаты углов фронтальной поверхности задаются относительно координат левого верхнего угла текущего графического окна.

void circle(int x, int y, int radius)

Выводит окружность заданного аргументом radius радиуса с центром, заданным координатами x и y. Координаты центра определяются относительно координат левого верхнего угла текущего графического окна. Для вывода окружности используется текущий цвет и только сплошная линия. Толщина линии (но не стиль!) может быть задана функцией settextstyle() (1 или 3 пиксела).

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

Область экрана внутри окружности может быть заполнена функцией floodfill() по текущей маске с использованием текущего цвета.

void drawpoly(int numpoints, int polypoints[])

"Соединяет" отрезками прямых линий текущего цвета и стиля точки (поли-

гон), 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. Текущая позиция не изменяется. Для получения замкнутых ломаных линий необходимо задать равными первую и последнюю точки выводимого полигона. Область экрана внутри полигона может быть заполнена с использованием текущего цвета и стиля заполнения функцией fillpoly().

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 (рис. 3.2): пиксельные координаты центра эллипса (x, y); начальный угол дуги (stangle); конечный угол дуги (endangle); радиус эллипса по горизонтали (xradius); радиус эллипса по вертикали (yradius). Функция выводит контур дуги или полный эллипс, используя непрерывную линию. Координаты центра задаются относительно координат левого верхнего угла текущего графического окна. Толщина линии (но не стиль!) может быть задана равной 1 или 3 пикселам функцией settextstyle(). Углы задаются в градусах и измеряются против хода часовой стрелки. Положению часовой стрелки 3 часа соответствует угол 0 градусов, 12 часов - 90 градусов, 9 часов - 180 градусов, 6 часов - 270 градусов. При задании stangle равным 0 градусов и endangle равным 359 градусов выводится полный эллипс. Текущая позиция при выводе дуги не изменяется. Функция автоматически корректирует координаты точек в соответствии с коэффициентом сжатия дисплея. Область экрана внутри эллипса может быть заполнена (функцией floodfill() или fillellipse() по текущей маске с использованием текущего цвета.

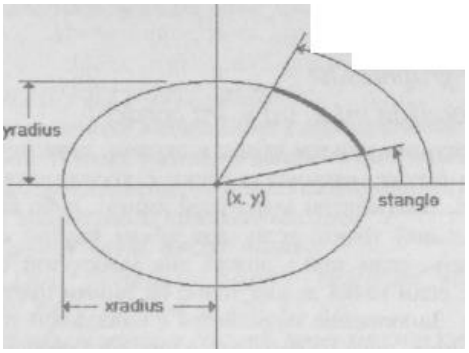

Рис. 3.2. Вывод эллиптических примитивов

void fillellipse(int x, int y, int xradius, int yradius)

Выводит эллипс, заполненный текущим стилем. Аргументы функции задают (см. рис. 3.2.): пиксельные координаты центра эллипса (x, y); радиус эллипса по горизонтали (xradius); радиус эллипса по вертикали (yradius). Функция выводит контур эллипса текущим цветом, устанавливаемым функцией setcolor(). Координаты центра задаются относительно координат левого верхнего угла текущего графического окна. Текущая позиция при выводе эллипса не изменяется. Функция автоматически корректирует координаты точек в соответствии с коэффициентом сжатия дисплея. Цвет и маска заполнения могут быть заданы с помощью функций setfillpattern() и setfillstyle().

При выполнении операции заполнения функция использует внутренний буфер библиотеки графики Turbo C для хранения промежуточных результатов. Если объем буфера будет недостаточным, функция установит в -6 внутренний код ошибки. Этот код доступен программе через обращение к функции graphresult().

void fillpoly(int numpoints, int *polypoints)

Выводит контур полигона, заданного numpoints точками. Координаты точек заданы парами, расположенными в массиве, на который ссылается polypoints. Функция соединяет первую и последнюю точки и заполняет область внутри полигона текущим стилем. Функция выводит контур полигона текущим цветом, устанавливаемым функцией setcolor(). Координаты точек задаются относительно координат левого верхнего угла текущего графического окна. Текущая позиция не изменяется. Цвет и маска заполнения могут быть заданы с помощью функций setfillpattern() и setfillstyle().

При выполнении операции заполнения функция использует внутренний буфер библиотеки графики для хранения промежуточных результатов. Если объем буфера будет недостаточным, функция установит в -6 внутренний код ошибки. Этот код доступен программе через обращение к функции graphresult().

void floodfill (int x, int y, int border)

Заполняет текущим стилем область экрана, ограниченную непрерывной линией с цветом border, начиная с точки с координатами (x, y). Функция заполняет область либо внутри замкнутой линии, либо вне ее. Это зависит от положения начальной точки: если она лежит внутри области, заполняется внутренняя область; если точка лежит вне замкнутой области, заполняется внешняя область; ес-

ли точка лежит точно на линии цвета border, заполнение не производится. Заполнение начинается с начальной точки и продолжается во всех направлениях, пока не встретится пиксел с цветом border. Цвет border должен отличаться от цвета заполнения, в противном случае будет заполнен весь экран. Цвет и маска заполнения могут быть заданы с помощью функций setfillpattern() и setfillstyle().

void pieslice(int x, int y, int stangle, int endangle, int radius)

Выводит контур кругового сектора и заполняет его внутреннюю область текущим стилем. Контур образован круговой дугой радиусом radius с координатами центра (x, y), проведенной, начиная от угла stangle до угла endangle, и радиусами, соединяющими центр с концевыми точками дуги. Дуга контура выводится текущим цветом, устанавливаемым функцией setcolor() всегда сплошной линией. Толщина (но не стиль!) дуговой линии равна 1 или 3 пикселам и задается функцией setlinestyle(). Стиль линии радиусов может быть любым и управляется функцией setlinestyle(). Особенностью рассматриваемой функции является то, что при задании любого другого стиля линии, отличного от сплошной линии (параметр linestyle в функции setlinestyle(), не равный 0), дуга сектора становится невидимой. Цвет и маска заполнения могут быть заданы с помощью функций setfillpattern() и setfillstyle().

При выполнении операции заполнения функция использует внутренний буфер библиотеки графики для хранения промежуточных результатов. Если объем буфера будет недостаточным, функция установит в -6 внутренний код ошибки. Этот код доступен программе через обращение к функции graphresult(). Углы stangle и endangle выводимой дуги задаются в градусах и измеряются против хода часовой стрелки. Положению часовой стрелки 3 часа соответствует угол 0 градусов, 12 часов - 90 градусов, 9 часов - 180 градусов, 6 часов - 270 градусов. При задании stangle равным 0 градусов и endangle равным 359 градусов выводится полная окружность.

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. Координаты углов задаются относительно координат левого верхнего угла текущего графического окна. Контур выводится линией текущего цвета и стиля. Цвет контура может быть установлен функцией setcolor(). Стиль линии может быть выбран или задан функцией setlinestyle().

```
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. Аргумент xradius задает радиус эллипса по горизонтали, а yradius - радиус эллипса по вертикали. При выводе сектора учитывается коэффициент сжатия, и эллиптическая дуга на экране геометрически корректна.

Перечисленными функциями исчерпывается список функций для вывода основных графических примитивов. Дополнительные графические примитивы могут быть построены из стандартных средств C++.

3.9. Предварительная подготовка к работе

1. Ознакомиться с организацией и функциональными возможностями различных типов видеосистем.
2. Ознакомиться с графическим режимом отображения информации на экран монитора и стандартными библиотечными функциями C, обслуживающими этот режим.

3.10. Порядок выполнения работы

1. Разработать программу для вывода на экран графика заданной функции.

Номер	Функция	Диапазон аргумента	
		Начало	Конец
1	$Sin^2(x/2)+Sqrt(x)$	$3\pi/2$	15π
2	$Sin^3(x/2)+Sqrt(x)$	$3\pi/2$	16π
3	$Sin^2(x/4)+Sqrt(x)$	$3\pi/2$	17π
4	$Cos^2(x/2)+Sqrt(x)$	$3\pi/2$	18π
5	$Cos^3(x/2)+Sqrt(x)$	$3\pi/2$	15π
6	$Cos^2(x/4)+Sqrt(x)$	$3\pi/2$	16π
7	$Sin^2(x)- Cos^2(x)$	$3\pi/2$	7π
8	$Sin^3(x)+ Cos^2(x)$	$3\pi/2$	8π
9	$Sin^2(x)+ Cos^3(x)$	$\pi/2$	5π
10	$Sin^3(x)+ Cos^3(x)$	$\pi/2$	6π
11	$Sin^2(x)- Cos^2(x)$	$\pi/2$	7π
12	$Sin^3(x)- Cos^2(x)$	$\pi/2$	8π
13	$Sin^2(x)- Cos^3(x)$	$\pi/2$	5π

Номер	Функция	Диапазон аргумента	
		Начало	Конец
14	$Sin^3(x)- Cos^3(x)$	π/2	6π
15	$Sin^2(x/2)$-Sqrt(x)	π/2	13π
16	$Sin^3(x/2)$-Sqrt(x)	π/2	12π
17	$Sin^2(x/4)$-Sqrt(x)	π	11π
18	$Cos^2(x/2)$-Sqrt(x)	π	10π
19	$Cos^3(x/2)$-Sqrt(x)	π	9π
20	$Cos^2(x/4)$-Sqrt(x)	π	8π

2. Произвести разметку осей и проставить истинные значения точек.

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

3.11. Содержание отчета

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

2. Алгоритмы и тексты отлаженных программ.

3. Структурная схема аппаратных средств, используемых при выполнении программы с необходимой степенью детализации содержимого блоков.

3.12. Контрольные вопросы