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

Лабораторная работа №3
ИССЛЕДОВАНИЕ ВИДЕОСИСТЕМЫ (ГРАФИЧЕСКИЙ РЕЖИМ)

Студентка гр. 6306 _____ Пустовойтова А. А.

Преподаватель _____ Бекенева Я. А.

Санкт-Петербург
2017

Краткие сведения

Видеорежимы и их характеристика

Интегральной характеристикой особенностей работы адаптера является совокупность поддерживаемых им режимов. Поведение адаптера в том или ином режиме является фактическим стандартом и полностью характеризует все особенности адаптера, доступные для программиста средства управления адаптером и т.п. Режимы принято нумеровать, начиная с нуля. Чем совершеннее видеоадаптер, тем больше режимов он поддерживает. Как правило, более совершенные адаптеры полностью совместимы со своими предшественниками и с точки зрения прикладной программы отображает информацию точно так же, как и его предшественник. Некоторые режимы работы видеоадаптеров описаны в табл. 2.1.

Табл. 2.1. Режимы работы видеоадаптеров

Режим	Тип	Размер шрифта	Максимальное число страниц	Разрешение		Адрес видеобуфера	Тип Видеоадаптера
				графика	Текст		
0,1	Текст	8x8	8	_	40x25	B8000h	CGA, EGA, VGA,MCGA
	Текст	8x14	8	_	40x25	B8000h	EGA,VGA
	Текст	8x16	8	-	40x25	B8000h	MCGA
	Текст	9x16	8	_	40x25	B8000h	VGA
2,3	Текст	8x8	4	-	80x25	B8000h	CGA
	Текст	8x8	8	-	80x25	B8000h	EGA,VGA
	Текст	8x8	8	_	80x43	B8000h	EGA
	Текст	8x8	8	_	80x50	B8000h	VGA
	Текст	8x14	8	-	80x25	B8000h	EGA,VGA
	Текст	8x16	8		80x25	B8000h	MCGA
	Текст	9x16	8	-	80x25	B8000h	VGA
4,5	Граф.	8x8	1	320x200	40x25	B8000h	CGA,EGA, VGA,AT&T MCGA
6	Граф.	8x8	1	640x200	80x25	B8000h	EGA,VGA
7	Текст	8x14	4		80x25	B0000h	EGA,VGA
Dh	Граф.	8x8	8	320x200	40x25	A0000h	EGA,VGA

Eh	Граф.	8x8	4	640x200	80x25	A0000h	EGA,VGA
Fh	Граф.	8x14	2	640x350	80x25	A0000h	EGA,VGA
10h	Граф.	8x14	2	640x350	80x25	A0000h	EGA,VGA
llh	Граф.	8x16	1	640x480	80x30	A0000h	MCGA, VGA
12h	Граф.	8x16	1	640x480	80x30	A0000h	VGA

При всем многообразии режимов работы видеоадаптеров их можно объединить в две группы: текстовые и графические. Переключение из текстового режима в графический и наоборот означает полное изменение логики работы видеоадаптера с видеобуфером. Если видеоадаптер включен в текстовый режим, он рассматривает экран как совокупность так называемых текселов (texel - Text Element) (рис. 2.1).

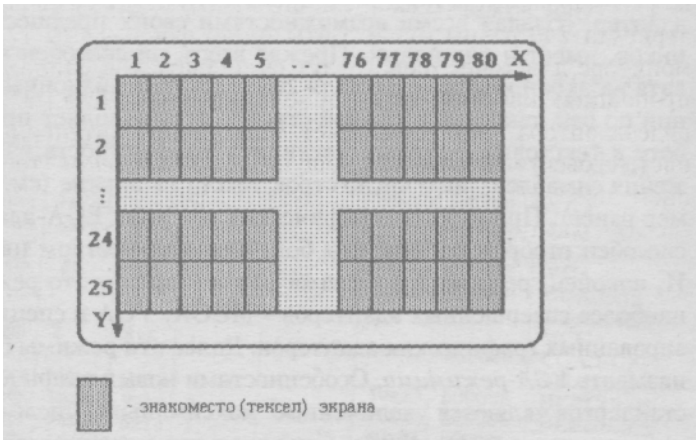


Рис. 2.1. Представление экрана в текстовых режимах "25 строк х 80 столбцов"

Каждому знакоместу экрана (текселу) в текстовом режиме соответствуют два байта памяти видеобуфера. Байт по четному адресу хранит ASCII-код символа, а следующий за ним байт по нечетному адресу кодирует особенности отображения символа на экране: цвет пикселов, из которых формируется очертание символа (Foreground Color), цвет всех остальных пикселов знакоместа или цвет фона символа (Background Color), мерцание символа и необходимость повышения яркости символа при отображении. Этот байт называется байтом атрибута. Закрепление битов байта атрибута приведено на рис. 2.2.

R, G, B -соответственно красный (Red), зеленый (Green), синий (Blue) цвета
(1 - цвет включен; 0 - цвет выключен)
Рис. 2.2. Назначение битов байта атрибута

Задавая различные числовые значения байту атрибута в видеобуфере, можно управлять цветом символов и цветом фона, на котором эти символы отображаются. Например, если значение байта атрибута равно 112, то выводится немерцающий символ черного цвета на сером фоне. Действительно, биты RGB цвета символа для данного кода атрибута равны нулю. Биты цвета фона равны 1, и на мониторе для точек фона будут смешиваться в необходимых пропорциях красный, синий и зеленый цвета. Для цветного видеоадаптера - это серый цвет. Повышение интенсивности цвета символа выполняется путем установки бита с номером 3 в 1. Светло-серый цвет - это белый цвет, поэтому на экране цветного монитора при работе видеоадаптера в текстовом режиме могут быть белые буквы, но не может быть белый фон. Например, символы, код атрибута которых в видеопамяти равен 15, будут отображаться белыми пикселами на черном фоне. В принципе, если задать цвета фона и символа одинаковыми, символы будут невидимыми, например красный символ на красном фоне (атрибут 0x44), что можно использовать в адаптерах, у которых мерцание символа с помощью бита 7 не реализовано.
Видеоадаптеры типов EGA и VGA имеют некоторые особенности использования бита интенсивности, которые будут рассмотрены несколько позже.
Видеопамять адаптера при работе в текстовых режимах доступна непосредственно из программы. Это значит, что любая ячейка видеобуфера может быть прочитана программой так же, как и обычная ячейка оперативной памяти. И как в обычную ячейку памяти, в видеобуфер возможна запись значений из программы. Адреса ячеек видеопамяти начинаются для разных типов адаптеров с разных границ, приведенных в табл. 2.1. Если адаптер работает в текстовых режимах "40 столбцов x 25 строк", то для хранения полного образа экрана (видеостраницы) требуется 25 x 40 x 2 = 2000 байт видеопамяти. В режимах "80 столбцов x 25 строк" видеостраница занимает уже 25 x 80 x 2 = 4000 байт.
Минимальная конфигурация видеоадаптера CGA имеет обычно 16K байт видеопамяти, что позволяет хранить 8 страниц текста в режимах 0 или 1 и 4 страницы в режимах 2 или 3.
Вывод на монитор содержимого видеобуфера происходит начиная с некоторого начального адреса, называемого смещением до видеостраницы. Страница 0 имеет нулевое смещение. Страница 1 в режиме "80 строк x 25 столбцов" начинается с адреса, смещенного на 4096 байт (l000h) относительно начального адреса видеопамяти, страница 2 - со смещения 8192 байт (2000h) и т.д. Если изменить значение смещения, произойдет переключение страницы, т.е. на экране возникнет образ другой страницы видеопамяти. Иногда переключение видеостраниц в текстовом режиме используется для реализации динамических изображений.
Видеоадаптер при работе в текстовом режиме периодически считывает содержимое ячеек видеобуфера и по коду символа и байту атрибута формирует пикселы, образующие в совокупности очертание символа и его фон. При этом байт символа служит индексом для входа в специальную таблицу - так называемую таблицу знакогенератора. Она содержит информацию, по которой видеоадаптер формирует пикселы для изображения того или иного символа. Число строк и столбцов в одной ячейке таблицы различно для различных

типов видеоадаптеров. Чем больше строк и столбцов использовано для символа, тем более качественно он изображается на экране.
Число знакомест в одной текстовой строке зависит от видеоадаптера и от режима его работы.
Переключение адаптера в один из графических режимов полностью изменяет логику работы аппаратуры видеосистемы. При работе в графическом режиме появляется возможность управлять цветом любой телевизионной точки экрана или пиксела. Число строк пикселов и число пикселов в каждой строке зависит от режима работы видеоадаптера. Таким образом, экран в графическом режиме представляет собой матрицу пикселов (рис. 2.3).

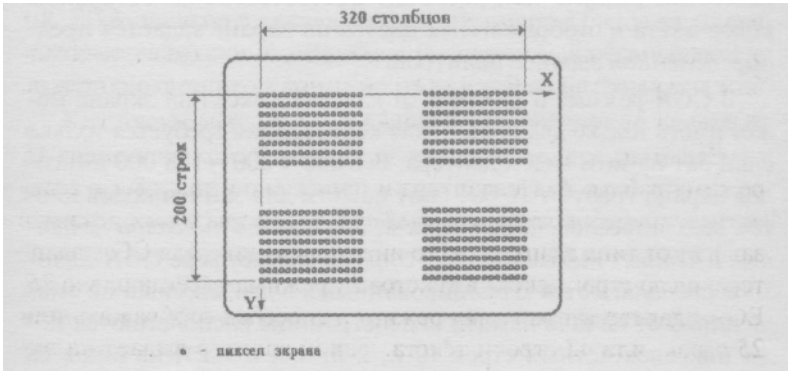


Рис. 2.3. Представление экрана в графических режимах 4,5 (320 столбцов х 200 строк)

Работа с нрафческой информацией

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 ATT400C3	320x200, палитра 0 320x200, палитра 1 320x200, палитра 2 320x200, палитра 3	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 определения 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 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.

Функции вывода 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 (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. 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 (фун-кцией floodfill() или fillellipse() по текущей маске с использованием текущего цвета.

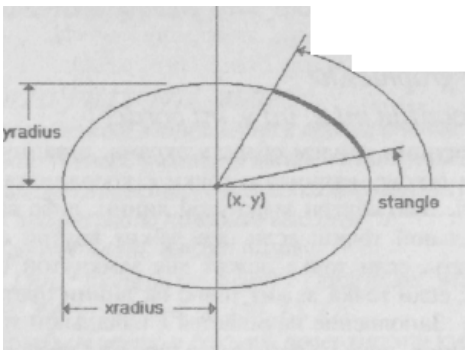


Рис. 3.2. Вывод эллиптических примитивов

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; если 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)

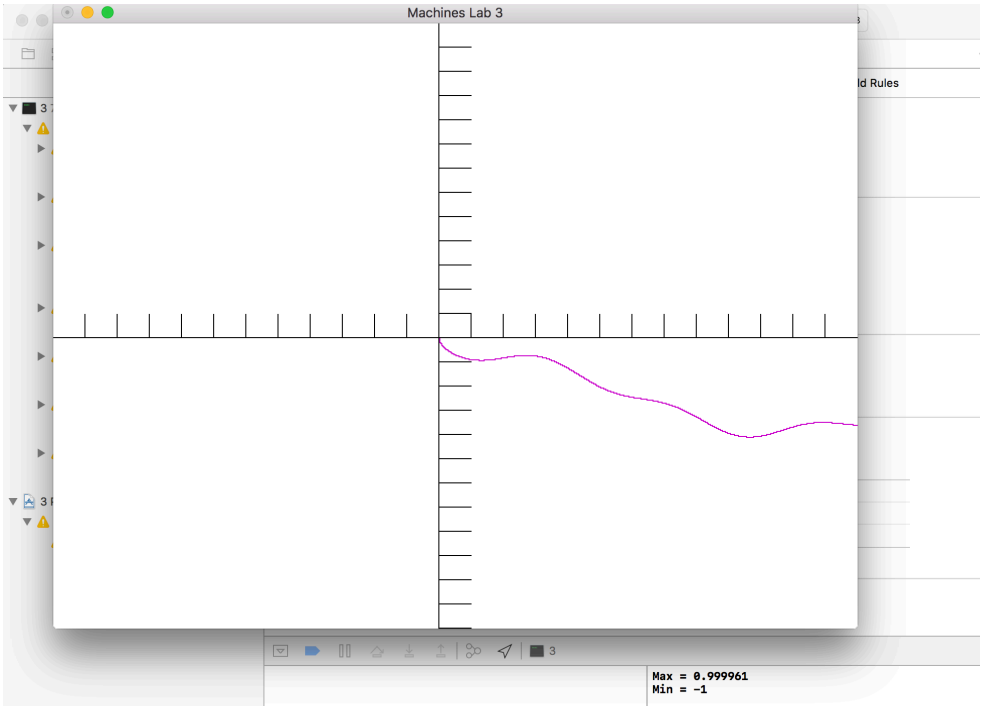
Работает аналогично функции pieslice(), за исключением того, что выводится не круговая, а эллиптическая дуга. Аргумент xradius задает радиус эллипса по горизонтали, а yradius - радиус эллипса по вертикали. При выводе сектора учитывается коэффициент сжатия, и эллиптическая дуга на экране геометрически корректна.

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

Задание

Написать программу так, чтобы в окно выводился график функции Sin3(x/2)-Sqrt(x) в диапазоне от pi/2 до 12pi. Произвести разметку, выводить максимум и минимум функции.

Пример запуска



Текст программы

Main.cpp

```
#include <iostream>
#include <GLUT/GLUT.h>
#include <cstdlib>
#include <cmath>

using namespace std;
const int X_COORD = 25;// X - размерность ] должны
const int Y_COORD = 25;// Y - размерность ] быть равными
const float ITERATIONS = 0.00005;// прорисовка графика (чем меньше тем лучше)

int x_off = X_COORD / 2;// начало
int y_off = Y_COORD / 2;// оси координат

//исходная функция
#define expr x

void drawgrid(float SERIF_OFFSET, float SERIF_DISTANCE) {


    //SERIF_DISTANCE - расстояние между засечками
    //SERIF_OFFSET - высота засечки
    glBegin(GL_LINES);
    //задаем цвета осей
    glColor3f(0.0, 0.0, 0.0);

    //рисуем координатные оси
    //горизонталь
    glVertex2f(0.0, Y_COORD / 2);
    glVertex2f(X_COORD, Y_COORD / 2);
    //засечки по горизонтали
    int p = X_COORD / 2;
    for(int i = X_COORD / 2; i < X_COORD; i += SERIF_DISTANCE, p -= SERIF_DISTANCE) {
        //засечка слева от оси y
        glVertex2f(i, Y_COORD / 2);
        glVertex2f(i, Y_COORD / 2 + SERIF_OFFSET);

        //засечка справа от оси y
        glVertex2f(p, Y_COORD / 2);
        glVertex2f(p, Y_COORD / 2 + SERIF_OFFSET);

        //i - координата засечек от центра >0
        //p -координата засечек от центра <0
    }
    //вертикаль
    int t = Y_COORD / 2;
    glVertex2f(X_COORD / 2, Y_COORD);
    glVertex2f(X_COORD / 2, 0.0);
    //засечки по вертикали
    for(int i = Y_COORD / 2; i < Y_COORD; i += SERIF_DISTANCE, t -= SERIF_DISTANCE) {
        glVertex2f(X_COORD / 2, i);
        glVertex2f(Y_COORD / 2 + SERIF_OFFSET, i);

        glVertex2f(X_COORD / 2, t);
        glVertex2f(Y_COORD / 2 + SERIF_OFFSET, t);
```

```
    }
    glEnd();
}

void drawfunc(float val1, float val2) {
    //рисуем график
    glBegin(GL_POINTS);
    float j = 0;
    glColor3f(0.8, 0.0, 0.8);
    for(float x = val1; x < val2; x += ITERATIONS) {
        //перерасчитываем координаты
        j = sin(x/2)*sin(x/2)*sin(x/2)-sqrt(x);
        glVertex2f(x_off + x, y_off + j);//не убирать x и y!! это оффсет по осям!
    }
    glEnd();
}

void funcinfo(float val1, float val2) {
    //информация о графике
    float max = -10, min = 10, coord;

    for(float x = val1; x <= val2; x++) {
        coord = sin(x);
        if (max < coord) max = coord;
        if (min > coord) min = coord;
    }
    cout << "Max = " << max << endl << "Min = " << min << endl;
    //    cout << "Sin(PI) = " << sin(M_PI) << endl;
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);


    drawgrid(1, 1);
    drawfunc(-M_PI/2, 12*M_PI);
    funcinfo(-M_PI/2, 12*M_PI);



    glutSwapBuffers();

    glFlush();
}

int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(800, 600);
    glutCreateWindow("Machines Lab 3");

    glClearColor(1.0, 1.0, 1.0, 1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    //пространство координат
    glOrtho(0.0, X_COORD, 0.0, Y_COORD, -1.0, 1.0);

    glutDisplayFunc(display);
    glutMainLoop();
}
```