

UNIT 12

Introduction to Graphics

C Programming (CACSI51)

12.1 Introduction to Graphics Programming

Every C program written so far has produced text-mode output — characters and numbers displayed line by line on the screen using `printf()`. C also allows a program to switch the display into a graphics mode, in which the screen is treated as a grid of individually addressable points called pixels. Once in graphics mode, a program can draw lines, shapes, patterns, and text at exact screen positions, which makes it possible to build simple drawings, charts, animations, and games.

On the (Turbo C / Turbo C++) compiler traditionally used for this course, graphics support is provided through a library called the BGI — Borland Graphics Interface. All the declarations needed for graphics programming are found in the header file `<graphics.h>`, and the actual drawing routines are supplied in driver files (commonly a file named `EGAVGA.BGI`) that must be available to the program at run time.

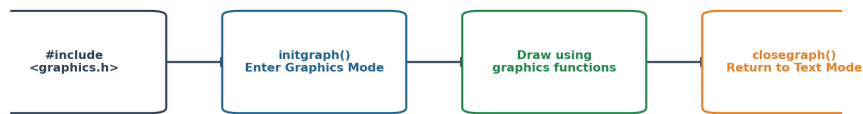
12.1.1 Why Use Graphics Mode?

- To display drawings, diagrams, and shapes rather than only plain text.
- To build simple visual programs such as bar charts, clocks, geometric designs, and basic animations.
- To control the exact color and position of individual points on the screen.
- To combine text and graphics on the same screen using special text-output functions.

Note: Graphics mode in this course refers to the classic DOS-style BGI graphics available with Turbo C / Turbo C++, which is still widely taught for understanding the fundamental ideas of pixel-based graphics programming.

12.2 Graphics Initialization

Before any drawing function can be used, the graphics system must be switched on. This process is called graphics initialization and is carried out by the function `initgraph()`. Once a program has finished drawing, the graphics system should always be switched off again using `closegraph()`, which returns the screen to normal text mode.



General sequence followed by every C program that uses graphics mode

Fig. 12.1 — The general sequence followed by a graphics program in C

12.2.1 The initgraph() Function

`initgraph(&gd, &gm, path)` takes three arguments: the address of a graphics driver variable, the address of a graphics mode variable, and the path (as a string) to the directory that holds the BGI driver files. Passing the constant `DETECT` as the driver value tells `initgraph()` to automatically detect the best graphics driver and mode supported by the system's display hardware.

```

#include <graphics.h>
#include <stdio.h>
#include <conio.h>

int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "C:\\\\TC\\BGI");

    /* ... drawing statements go here ... */

    getch(); /* pause so the drawing stays visible */
    closegraph(); /* shut down graphics, return to text mode */
    return 0;
}
  
```

Item	Purpose
<code>int gd = DETECT;</code>	<code>gd</code> (graphics driver) is auto-detected rather than fixed by the programmer
<code>int gm;</code>	<code>gm</code> (graphics mode) is filled in by <code>initgraph()</code> once the driver is detected
<code>"C:\\\\TC\\BGI"</code>	Path to the folder containing the .BGI driver files, e.g. <code>EGAVGA.BGI</code>
<code>getch()</code>	Holds the graphics screen on display until a key is pressed
<code>closegraph()</code>	Shuts down the graphics system and restores the original text screen

12.2.2 Checking for Initialization Errors

Graphics initialization can fail — for example, if the driver path is wrong, or the driver file is missing. The function `graphresult()` returns an error code that should be checked immediately after calling `initgraph()`, so the program can report a clear message instead of simply failing to display anything.

```
int errorcode = graphresult();
if (errorcode != grOk) {
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    exit(1);
}
```

12.3 Graphical Mode

A graphics mode defines the screen resolution (the number of pixels across and down) and the number of colors that can be displayed at the same time. Different graphics drivers and modes offer different combinations of resolution and color depth; higher resolution generally means sharper detail, while more available colors allow richer drawings.

Driver	Mode constant	Resolution	Colors
CGA	CGAC0 – CGAC3	320 × 200	4
EGA	EGALO / EGAHI	640 × 200 / 640 × 350	16
VGA	VGALO / VGAMED / VGAHI	640 × 200 to 640 × 480	16
ATT400	ATT400C0 – ATT400C3	320 × 200 to 640 × 400	2 – 4

Note: On modern systems, DETECT together with a VGA-capable driver typically selects VGAHI automatically, giving 640 x 480 resolution with 16 colors — the mode most commonly used in this course.

12.3.1 The Screen Coordinate System

Once in graphics mode, every point on the screen is identified by an (x, y) pair. Unlike ordinary mathematical graphs, the origin (0, 0) lies at the top-left corner of the screen: the x-value increases toward the right, and the y-value increases downward toward the bottom of the screen.

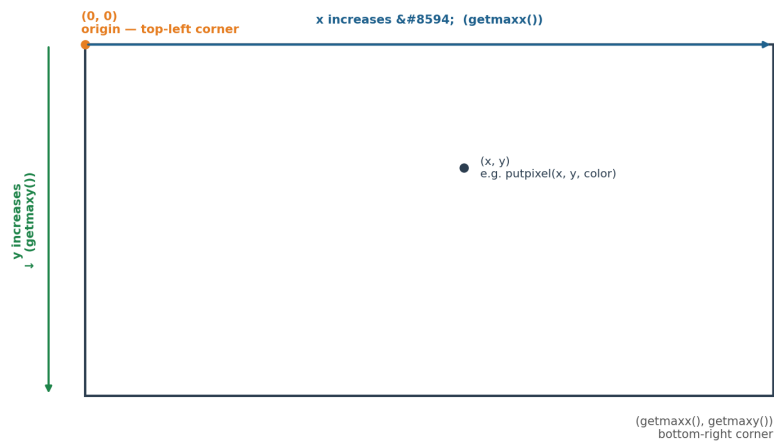


Fig. 12.2 — The pixel coordinate system used in graphics mode

The functions `getmaxx()` and `getmaxy()` return the largest valid x and y coordinate for the current mode, which makes it possible to write drawings that automatically adjust to the size of the screen rather than using fixed numbers.

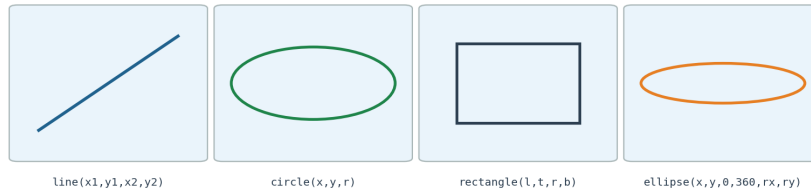
```
int mx = getmaxx(); /* rightmost pixel column */
int my = getmaxy(); /* bottommost pixel row */
rectangle(0, 0, mx, my); /* border around the whole screen */
```

12.3.2 Changing Mode and Restoring the Screen

Function	Purpose
<code>setgraphmode(mode)</code>	Switches to a specific graphics mode during a program
<code>restorecrtmode()</code>	Temporarily restores the original text screen without ending graphics
<code>getgraphmode()</code>	Returns the graphics mode currently in use
<code>cleardevice()</code>	Clears the entire graphics screen, ready for a fresh drawing

12.4 Graphical Functions

Once graphics mode is active, `<graphics.h>` provides a large family of functions for drawing shapes, setting colors, and combining text with graphics. A few of the most frequently used ones are introduced below.



A few of the basic shape-drawing functions available once graphics mode is initialized

Fig. 12.3 — A few of the basic shape-drawing functions

12.4.1 Drawing Basic Shapes

Function	Description
line(x1,y1,x2,y2)	Draws a straight line between two points
circle(x,y,r)	Draws a circle of radius r centered at (x, y)
rectangle(l,t,r,b)	Draws a rectangle using its left, top, right, and bottom coordinates
ellipse(x,y,s,e,rx,ry)	Draws an elliptical arc from angle s to angle e
arc(x,y,s,e,r)	Draws a circular arc from angle s to angle e
drawpoly(n,points[])	Draws a polygon joining n given points
putpixel(x,y,color)	Plots a single pixel at (x, y) in the given color
getpixel(x,y)	Returns the color of the pixel currently at (x, y)

Example 12.1 — Draw a circle and a rectangle

Initialize graphics mode, draw a circle and a rectangle on the screen, then wait for a key press before closing graphics mode.

```
#include <graphics.h>
#include <conio.h>

int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "C:\\\\TC\\\\BGI");

    circle(150, 150, 60);
    rectangle(250, 90, 400, 210);

    getch();
    closegraph();
    return 0;
}
```

12.4.2 Filled Shapes and Fill Patterns

Shapes can be filled with a color or pattern instead of being drawn as an outline only. `setfillstyle(pattern, color)` chooses the fill pattern and color to be used, and `floodfill(x,y,border)` fills an enclosed region starting from a given interior point, stopping at the specified border color. Ready-made filled shapes are also available directly, such as `bar()` for a filled rectangle and `pieslice()` for a filled pie-shaped wedge.

Function	Description
<code>setfillstyle(pattern,color)</code>	Selects the fill pattern (e.g. <code>SOLID_FILL</code>) and its color
<code>floodfill(x,y,border)</code>	Fills a bounded region starting at (x, y) up to the border color
<code>bar(l,t,r,b)</code>	Draws a filled rectangle
<code>bar3d(l,t,r,b,depth,top)</code>	Draws a filled 3-D-style rectangular bar
<code>pieslice(x,y,s,e,r)</code>	Draws and fills a pie-slice wedge from angle s to angle e

Example 12.2 — Draw a filled bar chart

Draw three filled bars of different heights, side by side, to represent a simple bar chart.

```
#include <graphics.h>
#include <conio.h>

int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "C:\\\\TC\\BGI");

    setfillstyle(SOLID_FILL, RED);
    bar(100, 300, 150, 150);

    setfillstyle(SOLID_FILL, GREEN);
    bar(200, 300, 250, 100);

    setfillstyle(SOLID_FILL, BLUE);
    bar(300, 300, 350, 200);

    getch();
    closegraph();
    return 0;
}
```

Explanation

setfillstyle() is called before each bar() to change the fill color used for that bar. bar(l,t,r,b) fills the rectangle from (l,t) to (r,b); since y increases downward, a smaller top value for the same bottom value produces a taller bar.

12.4.3 Setting Colors

Function	Purpose
setcolor(color)	Sets the color used for lines, shape outlines, and text
setbkcolor(color)	Sets the background color of the screen
getcolor()	Returns the color currently selected for drawing
getmaxcolor()	Returns the highest valid color number for the current mode

Note: Colors such as RED, GREEN, BLUE, YELLOW, WHITE, and BLACK are predefined constants declared in graphics.h, corresponding to numeric color codes (0-15 in a 16-color mode).

12.4.4 Combining Text with Graphics

Ordinary printf() cannot be used to place text at an exact pixel position on the graphics screen. Instead, <graphics.h> provides outtextxy(x, y, string), which writes a string so that its top-left corner begins at pixel position (x, y), and settextstyle(), which selects the font, direction, and size used for such text.

```
settextstyle(DEFAULT_FONT, HORIZ_DIR, 2);
outtextxy(50, 50, "Simple Bar Chart");
```

Key Takeaways

Unit 12 — Summary

- ✓ Graphics mode treats the screen as a grid of pixels, each identified by an (x, y) coordinate.
- ✓ `<graphics.h>` provides the BGI (Borland Graphics Interface) functions used for graphics programming in C.
- ✓ `initgraph()` switches the screen into graphics mode; `closegraph()` returns it to text mode.
- ✓ `graphresult()` should be checked after `initgraph()` to catch initialization errors.
- ✓ A graphics mode fixes the screen resolution and number of available colors; `getmaxx()/getmaxy()` report the current screen limits.
- ✓ The origin (0,0) is at the top-left corner; y increases downward, unlike a standard math graph.
- ✓ Basic functions such as `line()`, `circle()`, `rectangle()`, and `ellipse()` draw outline shapes; `setfillstyle()`, `bar()`, and `floodfill()` produce filled shapes.
- ✓ `setcolor()/setbkcolor()` control drawing and background colors; `outtextxy()` places text at an exact pixel position.

— End of Unit 12 —