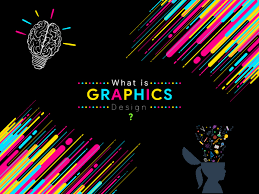
GRAPHICS DEVICE INTERFACE

The Graphics Device Interface (GDI) is a crucial component of Microsoft Windows, responsible for displaying graphics on video displays and printers.



GDI plays a pivotal role in both user applications and the Windows operating system itself, handling the visual rendering of elements such as menus, scroll bars, icons, and mouse cursors.

This chapter provides a fundamental understanding of GDI, focusing on the basics of drawing lines and filled areas.

This foundational knowledge will serve as a stepping stone for subsequent chapters that delve into more advanced GDI concepts, including bitmap support, metafiles, and formatted text.

The GDI Philosophy

The Graphics Device Interface (GDI) is a fundamental component of Microsoft Windows, responsible for rendering graphics on video displays and printers.

GDI functions are exported from the dynamic-link library GDI32.DLL.



In Windows 98, GDI32.DLL utilizes the 16-bit dynamic-link library GDI.EXE for the implementation of many of its functions.

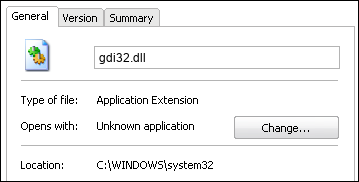
However, in Windows NT, GDI.EXE is only employed for 16-bit programs.

These dynamic-link libraries interact with device drivers for the video display and any connected printers.

The video driver interfaces with the video display hardware, while the printer driver translates GDI commands into codes or commands that the respective printers can interpret.

Consequently, different video display adapters and printers require specific device drivers.

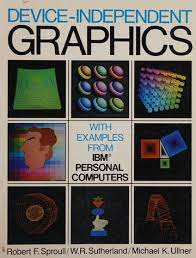
While Windows NT used a separate 16-bit dynamic-link library, GDI.EXE, for the implementation of GDI functions in 16-bit programs, this functionality has been integrated into GDI32.DLL in Windows 10 and 11.



Device-Independent Graphics

GDI is designed to support device-independent graphics, enabling Windows applications to function seamlessly on any compatible graphics output device.

This goal is achieved by providing mechanisms that isolate programs from the unique characteristics of different output devices.



Raster vs. Vector Devices

Graphics output devices can be categorized into two main types: raster devices and vector devices.

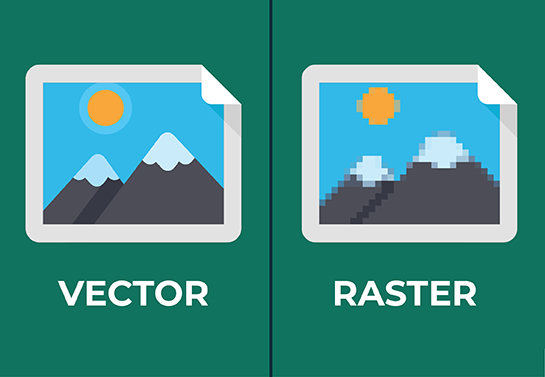
Raster devices, which include video display adapters, dot-matrix printers, and laser printers, represent images as a rectangular pattern of dots.

Vector devices, primarily limited to plotters these days, generate images using lines.



GDI as a High-Level Interface

Traditional computer graphics programming often relies solely on vectors, introducing an abstraction layer between the program and the hardware.



While output devices utilize pixels for graphics representation, the program doesn't directly interact with the hardware in terms of pixels.

The Windows GDI can be used as both a high-level vector drawing system and a relatively low-level pixel manipulation tool.

In this sense, GDI parallels C's position among programming languages. C is renowned for its portability across different operating systems and environments, while also allowing programmers to perform low-level system functions often inaccessible in other high-level languages.

Just as C is sometimes considered a "high-level assembly language," GDI can be viewed as a high-level interface to the graphics device hardware.

Coordinate Systems

Windows employs a pixel-based coordinate system by default.

Traditional graphics languages typically use a "virtual" coordinate system with horizontal and vertical axes ranging from 0 to 32,767, for instance.

While some graphics languages restrict pixel coordinates, Windows GDI allows using both systems, along with additional coordinate systems based on physical measurements.

Programmers' Control

Programmers can opt for a virtual coordinate system to maintain a level of abstraction from the hardware or utilize the device coordinate system for closer hardware interaction.

Some programmers argue that using pixels signifies a departure from device independence.

However, as discussed earlier, this is not entirely true.

The key lies in using pixels in a device-independent manner.

This requires the graphics interface language to provide mechanisms for a program to determine the hardware characteristics of the device and make appropriate adjustments.

For instance, in the SYSMETS programs, the pixel size of a standard system font character was used to space text on the screen.

This approach allowed the programs to adapt to different display adapters with varying resolutions, text sizes, and aspect ratios.

Other methods for determining display sizes will be introduced in subsequent chapters.

Monochrome Displays

In the early days of Windows, many users ran the operating system with a monochrome display.

This meant that the display could only display two colors: black and white.

As a result, GDI was designed to allow programmers to write programs without having to worry about color.

Windows would automatically convert any colors used in the program to shades of gray.

Color Displays

Even in more recent years, some users, such as laptop users, were restricted to gray shades.

However, with the increasing affordability of color displays, the number of users with color displays has grown significantly.

Today, video displays used with Windows 98 have different color capabilities, ranging from 16 colors to "true color" with millions of colors.

Inkjet vs. Laser Printers

Inkjet printers have brought low-cost color printing to the masses, but many users still prefer black-only laser printers for high-quality output.

It is possible to use these devices blindly, but your program can also determine how many colors are available on the particular output device and take best advantage of the hardware.

Device Dependencies

Just as you can write C programs that have subtle portability problems when they run on other computers, you can also inadvertently let device dependencies creep into your Windows programs.

That's part of the price of not being fully insulated from the hardware.

Animation Support

GDI is generally a static display system with only limited animation support.

If you need to write sophisticated animations for games, you should explore Microsoft DirectX, which provides the support you'll need.

In summary, it is important to consider the following when writing graphics programs for Windows:

* The color capabilities of the display device.
* The type of output device (e.g., printer).
* The limitations of GDI.

By taking these factors into account, you can write programs that are more portable and perform better on a wider range of devices.

Additional Points

GDI is a powerful tool for creating graphics programs for Windows.

GDI is designed to be device independent, so that programs can run on a variety of hardware.

GDI has some limitations, such as its limited animation support.

If you need to write sophisticated animations, you should explore Microsoft DirectX.

TYPES OF GDI FUNCTION CALLS

The GDI function calls can be broadly categorized into the following groups:

Device Context Management:

BeginPaint and EndPaint: These functions are part of the USER module and are used to obtain and release a device context during the WM\_PAINT message.

GetDC and ReleaseDC: These functions are used to obtain and release a device context during other messages.

Device Context Information Access:

GetTextMetrics: This function retrieves information about the dimensions of the currently selected font in the device context.

DEVCAPS1: This program obtains more general device context information.

Drawing Functions:

TextOut: This function displays text in the client area of the window.

Other drawing functions: GDI provides functions for drawing lines, filled areas, and other graphical elements.

Device Context Attribute Management:

SetTextColor: This function specifies the color of text drawn using TextOut and other text output functions.

SetTextAlign: This function informs GDI that the starting position of the text string in TextOut should be the right side of the string rather than the left.

GDI Object Manipulation:

CreatePen, CreatePenIndirect, and ExtCreatePen: These functions create logical pens, which define the attributes of lines drawn using GDI.

Pen Selection and Deselection: Pens are selected into the device context using their handle and deselected when no longer needed. Destroying pens is crucial to release allocated memory.

Brushes, Fonts, Bitmaps: GDI objects also include brushes for filling enclosed areas, fonts for text rendering, and bitmaps for image display.

These categories provide a comprehensive overview of the GDI function calls and their respective purposes.

GDI primitives:

Lines and Curves

Lines are the fundamental building blocks of any vector graphics drawing system. GDI supports a variety of line types, including straight lines, rectangles, ellipses (including circles), arcs (partial curves on an ellipse's circumference), and Bezier splines. If you need to draw a different type of curve, you can approximate it using a polyline, a series of very short lines that define the curve's shape. GDI renders lines using the current pen selected in the device context.

Filled Areas

When a series of lines or curves encloses an area, you can instruct GDI to fill that area with the current GDI brush object. This brush can be a solid color, a pattern (such as a series of horizontal, vertical, or diagonal hatch marks), or a bitmapped image that is replicated vertically or horizontally within the area.

Bitmaps

A bitmap, also known as a raster image, is a rectangular array of bits that correspond to the pixels of a display device. Bitmaps are the foundation of raster graphics and are commonly used for displaying complex images, including real-world scenes, on the video display or printer. Bitmaps are also employed for displaying small images that require rapid rendering, such as icons, mouse cursors, and toolbar buttons. GDI supports two types of bitmaps: device-dependent bitmaps, which are GDI objects, and device-independent bitmaps (DIBs), which were introduced in Windows 3.0 and can be stored in disk files. Bitmaps will be discussed in detail in Chapters 14 and 15.

Text

Text, unlike other aspects of computer graphics, is not entirely mathematical; it is rooted in centuries of traditional typography, considered an art form by many typographers and design enthusiasts. Consequently, text is often the most complex component of any computer graphics system, but it is also the most crucial aspect, assuming literacy remains the norm. Among the largest data structures in Windows are those used to define GDI font objects and retrieve font information. Starting with Windows 3.1, GDI began supporting TrueType fonts, which are based on filled outlines that can be manipulated using other GDI functions. For backward compatibility and low memory requirements, Windows 98 continues to support older bitmap-based fonts. Fonts will be discussed in detail in Chapter 17.

These four categories encompass the primary GDI primitives and provide a solid foundation for creating a wide range of graphical elements.