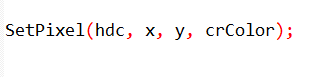
DRAWING LINES AND DOTS

Chapter 5 Part 2

SetPixel Function

The SetPixel function is a fundamental building block for creating graphics in Windows GDI. It allows you to set the color of a specific pixel at a specified x and y coordinate. The syntax is as follows:



hdc: A handle to the device context (DC) that represents the drawing surface.

x: The x-coordinate of the pixel to be set.

y: The y-coordinate of the pixel to be set.

crColor: A COLORREF value representing the desired color of the pixel.

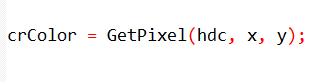
When called, SetPixel sets the specified pixel to the given color.

If the specified color cannot be represented on the video display, the function sets the pixel to the nearest pure non-dithered color and returns that value.

GetPixel Function

The GetPixel function is another essential tool for working with pixels in Windows GDI.

It retrieves the color of a specific pixel at a specified x and y coordinate. The syntax is as follows:



hdc: A handle to the DC that represents the drawing surface.

x: The x-coordinate of the pixel to retrieve the color from.

y: The y-coordinate of the pixel to retrieve the color from.

The GetPixel function returns a COLORREF value representing the color of the specified pixel.

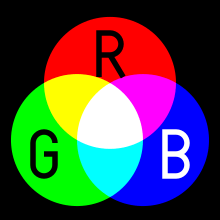
Limitations of SetPixel and GetPixel

While SetPixel and GetPixel provide direct access to individual pixels, they are not commonly used for complex graphics operations. This is primarily due to performance considerations.

Performance Overhead: Drawing complex shapes using SetPixel involves calling the function repeatedly for each pixel, which can be inefficient. Higher-level GDI functions, such as LineTo and Polyline, are optimized for efficient line drawing and utilize specialized hardware acceleration when available.



Device-Dependent Colors: COLORREF values represent colors in a device-dependent manner. Using SetPixel and GetPixel directly can lead to color discrepancies between different display devices.



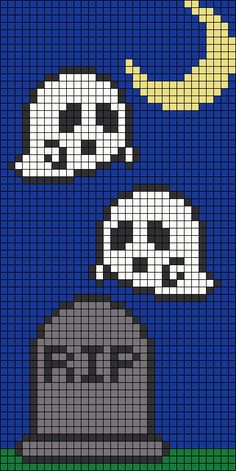
Alternative Graphics Approaches

In general, it is recommended to use higher-level GDI functions whenever possible.

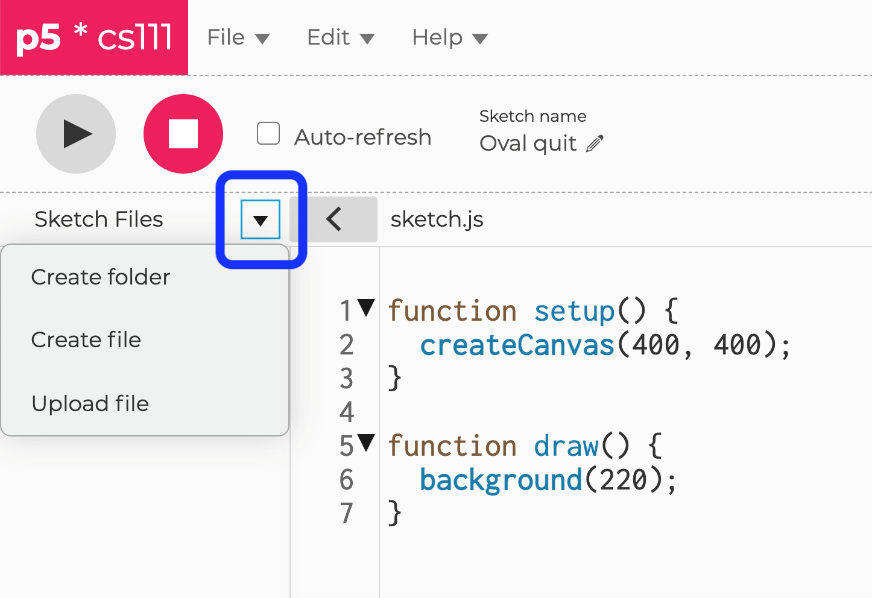
These functions provide better performance, hardware acceleration, and device-independent color handling.

For specialized cases where direct pixel manipulation is required, there are alternative approaches that offer more efficiency and flexibility.

Raster Operations: GDI provides raster operations (Rops) that allow for efficient manipulation of pixel patterns. These operations can be combined with SetPixel and GetPixel to achieve more complex graphics effects.



Custom Drawing Functions: Developers can create their own drawing functions that employ optimized algorithms and utilize hardware acceleration when available. This approach can be particularly beneficial for specialized graphics tasks.

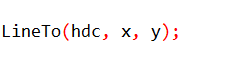


Line Drawing Functions in Windows GDI

Windows GDI provides a variety of functions for drawing straight lines. These functions offer different levels of flexibility and control over the line drawing process.

LineTo Function

The LineTo function draws a single straight line from the current pen position to the specified endpoint. The syntax is as follows:



hdc: A handle to the device context (DC) that represents the drawing surface.

x: The x-coordinate of the endpoint of the line.

y: The y-coordinate of the endpoint of the line.

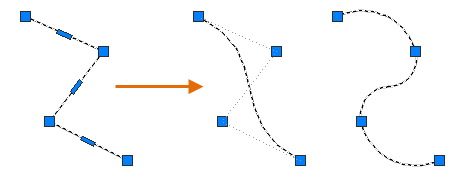
After calling LineTo, the current pen position is updated to the endpoint of the drawn line.

Polyline and PolylineTo Functions

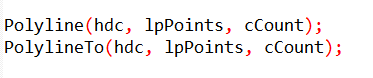
Polyline provides a more efficient approach for drawing a series of connected lines.

It takes an array of POINT structures, each representing a vertex of the polyline, and the number of points in the array.

Polyline draws lines connecting the specified points, effectively creating a continuous polyline.



The Polyline and PolylineTo functions draw a series of connected straight lines. Polyline defines an open polyline, while PolylineTo defines a closed polyline. The syntax for both functions is as follows:



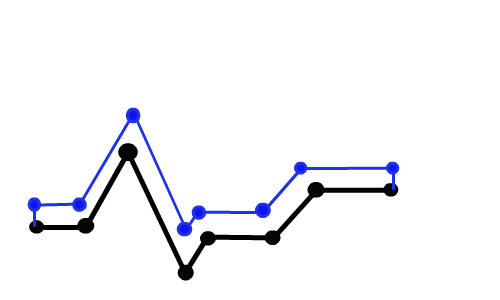
hdc: A handle to the DC that represents the drawing surface.

lpPoints: A pointer to an array of POINT structures, where each POINT structure specifies an x-coordinate and a y-coordinate for a vertex of the polyline.

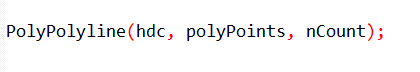
cCount: The number of vertices in the polyline.

PolylineTo is similar to Polyline, but it utilizes the current position of the DC as the starting point of the polyline. It then draws lines connecting the remaining points in the provided array, updating the current position to the endpoint of the last line drawn.

PolyPolyline Function



The PolyPolyline function draws multiple polylines. The syntax is as follows:



hdc: A handle to the DC that represents the drawing surface.

polyPoints: A pointer to an array of POLYLINE structures, where each POLYLINE structure specifies a polyline using its lpPoints member and its cCount member.

nCount: The number of polylines in the array.

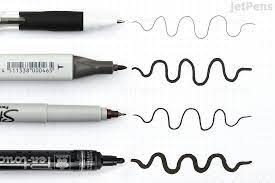
Factors Affecting Line Appearance

Five device context attributes influence the appearance of lines drawn using these functions:

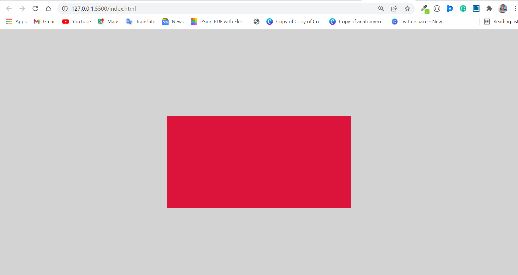
Current pen position: This attribute applies to LineTo, PolylineTo, PolyBezierTo, and ArcTo functions. It determines the starting point of the line.



Pen: The pen defines the style and attributes of the line, including its width, color, and pattern.



Background mode: This attribute determines how the background color is handled when drawing lines. OPAQUE mode fills the background with the specified color, while TRANSPARENT mode allows the underlying background to show through.



Background color: This attribute specifies the color used to fill the background in OPAQUE mode.



Drawing mode: This attribute determines how source pixels are combined with destination pixels when drawing lines. It affects how the line color is blended with the background color.

DRAWING STRAIGHT LINES WITH MOVETOEX AND LINETO

Drawing straight lines in Windows GDI involves two primary functions: MoveToEx and LineTo.

These functions work together to define the starting point and endpoint of the line.

MoveToEx Function

The MoveToEx function sets the current position of the device context (DC).

This position serves as the starting point for subsequent drawing operations, including line drawing. The syntax is as follows:



hdc: A handle to the DC that represents the drawing surface.

xBeg: The x-coordinate of the starting point.

yBeg: The y-coordinate of the starting point.

NULL: A placeholder for the previous current position value, which is not used in most cases.

The MoveToEx function doesn't actually draw anything; it simply updates the DC's current position.

This position remains the starting point until it is explicitly changed by another MoveToEx call or by certain other GDI functions.

MoveToEx and LineTo work together to draw individual lines.

MoveToEx sets the current position of the device context (DC), which defines the starting point of the line.

LineTo then draws a straight line from the current position to the specified endpoint. The current position is updated to the endpoint after drawing the line.

LineTo Function

The LineTo function draws a straight line from the current position of the DC to the specified endpoint. The syntax is as follows:



hdc: A handle to the DC that represents the drawing surface.

xEnd: The x-coordinate of the endpoint.

yEnd: The y-coordinate of the endpoint.

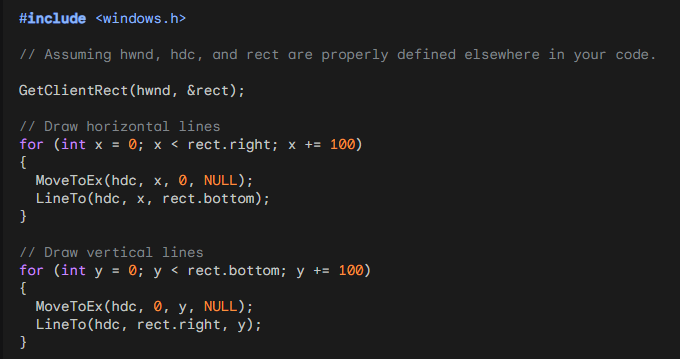
The LineTo function draws a line from the current position, which was previously set by the MoveToEx call, to the specified endpoint.

Once the line is drawn, the current position is updated to the endpoint.

Drawing a Grid with MoveToEx and LineTo

The provided code demonstrates how to draw a grid in the client area of a window using MoveToEx and LineTo.

It iterates through the client area coordinates, alternately calling MoveToEx to set the current position and then calling LineTo to draw a line.



This code snippet effectively draws a series of vertical and horizontal lines spaced 100 pixels apart within the client area of the window.

Windows 10 and 11 both use the full 32-bit values for coordinates, providing a wider range of representable positions within the drawing surface. This allows for more precise positioning of graphical elements and enables the creation of larger and more detailed graphics.

With the transition to 32-bit and later 64-bit operating systems, the coordinate range has expanded significantly, providing greater flexibility for graphics programming.

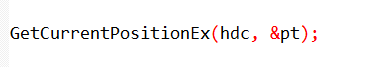
This broader range of coordinates is particularly beneficial for applications that require precise positioning of graphical elements, such as CAD software, architectural design tools, and high-resolution image editing programs.

It also allows for the creation of larger and more complex graphics without encountering limitations due to coordinate values.

Both Windows 10 and 11 fully utilize the 32-bit range for coordinate values, eliminating the limitations faced in earlier versions of the operating system. This provides a more versatile and expansive canvas for graphics applications.

Retrieving the Current Position

If you need to retrieve the current position of the DC, you can use the GetCurrentPositionEx function:



This function stores the current position in the provided POINT structure.

Choosing the Right Function

The choice between MoveToEx, LineTo, Polyline, and PolylineTo depends on the specific drawing requirements:

Individual Lines: Use MoveToEx followed by LineTo for drawing a single line.

Series of Connected Lines: Use Polyline for drawing a sequence of connected lines, especially when dealing with a large number of points.

Starting from Current Position: Use PolylineTo to draw a polyline starting from the current position of the DC.

Application in Drawing a Sine Wave

The SINEWAVE program demonstrates the use of Polyline to draw a sine wave.

It calculates a series of points representing the sine function and then invokes Polyline to connect these points, creating the smooth curve of the sine wave.

*Code found in chapter 5 (sinwave folder)*

*Preprocessor Directives and Constants*

The code begins by including the necessary header files: windows.h for Windows API functions and math.h for mathematical operations.

It then defines two constants: NUM, which represents the number of points in the sine wave (1000 in this case), and TWOPI, which represents twice the value of pi (used for calculating sine wave values).

*WinMain Function*

The WinMain function serves as the entry point for the application. It performs the following tasks:

Registers the Window Class: It defines the window class properties using WNDCLASS structure, including its name, style, and associated functions. This class determines the behavior and appearance of the application's window.

Creates the Window: It creates the application window using the CreateWindow function, specifying its name, class, initial position and size, and instance handle.

Shows the Window: It displays the created window using the ShowWindow function, making it visible to the user.

Updates the Window: It updates the window's contents using UpdateWindow, ensuring the sine wave is drawn correctly.

Message Loop: It enters a message loop using GetMessage and DispatchMessage functions. This loop continuously processes messages from the system and directs them to the appropriate handler functions.

Returns Exit Code: Finally, it returns the exit code from the message loop, indicating the application's termination status.

*WndProc Function*

The WndProc function handles various messages sent to the application window. It processes the following messages:

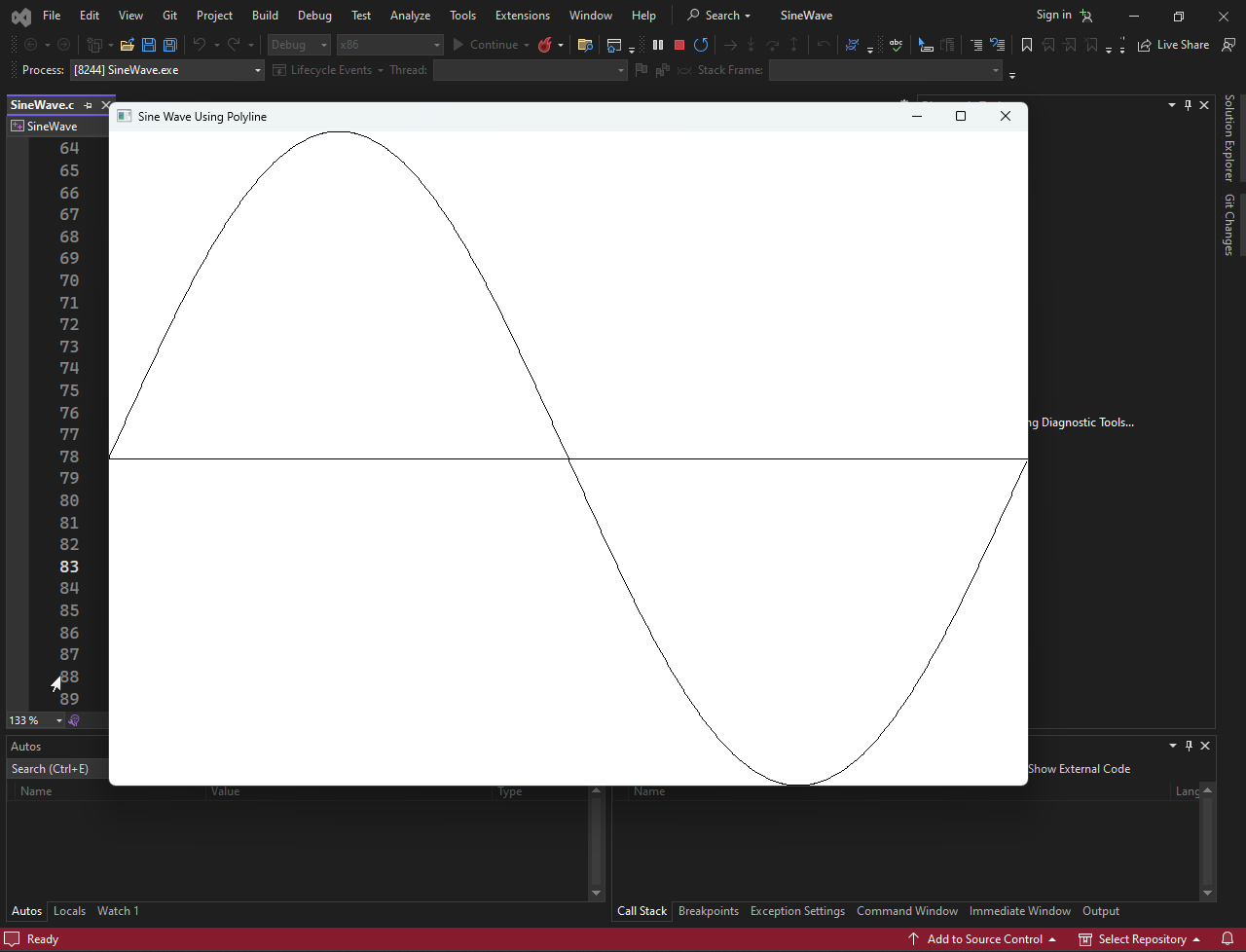
WM\_SIZE: This message is received when the window's size changes. The function stores the new client area width and height in cxClient and cyClient variables for later use.

WM\_PAINT: This message is received when the window needs to be repainted. The function performs the following tasks:

* Obtains Device Context: It retrieves the device context (DC) using BeginPaint.
* Draws Axis Line: It draws a horizontal line across the middle of the client area using MoveToEx and LineTo functions.
* Calculates Sine Wave Values: It iterates over the NUM points, calculating the sine wave value (scaled to fit the client area's height) for each point.
* Sets Point Coordinates: For each point, it sets the x-coordinate to its corresponding position along the client area's width and the y-coordinate to the calculated sine wave value.
* Draws Sine Wave: It draws the entire sine wave using a single Polyline call, providing the array of POINT structures containing the calculated coordinates.

WM\_DESTROY: This message is received when the window is destroyed. The function posts a quit message to terminate the application.

Default Window Procedure: For any unhandled messages, it calls the DefWindowProc function, allowing the default message handling mechanism to take over.



*Conclusion*

The provided code effectively demonstrates the use of Polyline to draw a smooth sine wave within the client area of a window.

It highlights the efficiency of Polyline compared to calling LineTo multiple times and showcases the application of mathematical functions in graphics programming.