Paint Application with ICBYTES Library and Real-Time Drawing Performance with Parallel Processing Techniques

Abstract

This project presents the graphical processing structure, multi-threading architecture, buffer canvas usage and performance optimizations of a drawing application developed in C++ and Visual Studio 2022. Drawing operations are managed with the **ICBYTES** library and a multi-threading supported structure provides real-time speed improvement. During drawing, a temporary canvas (tempCanvas) is used to make temporary drawings on the screen and then transferred to the main canvas (m). This method ensures graphical consistency.

In addition, a transparency-assisted (4-channel RGBA) canvas structure was implemented, resulting in color combinations and layer-based visuals during drawing. Multi-threaded drawing algorithms were optimized using parametric formulas for Bresenham line algorithm, Midpoint Circle algorithm and ellipse drawing. Features such as real-time color palette, pencil thickness, eraser and drawing of different geometric shapes have been added to facilitate user interaction.

Keywords: Computer Graphics, ICBYTES Library, C++ Graphics Programming, Multi-threading in Graphics, Bresenham's Algorithm, Graphical User Interface (GUI), Real-time Drawing, Geometric Shape Rendering, Interactive Drawing Application, Parallel Computing, Mutex Synchronization

Introduction

Real-time drawing applications in computer graphics are critical to the user experience. In traditional drawing applications, each pixel needs to be manipulated directly, which can result in latency and performance losses. Especially high resolution environments and complex geometric operations can negatively affect user interaction by increasing data processing times.

This project focuses on optimizing geometric shapes and freehand drawing operations using **C++** and the **ICBYTES** library. By implementing a multi-threaded architecture with parallel processing techniques, it is aimed to speed up the drawing process.

Purpose

This project aims to develop a high-performance and multi-threaded drawing application.The main objectives are:

- Providing instant feedback to the user by using a temporary canvas
- Improving drawing performance with multi-threaded algorithms
- Create a flexible structure that supports drawing different shapes
- Create layer-based drawings with a canvas that supports transparency (alpha channel)

Technologies and Methods

Programming Language and Development Environment:

- **C++**
- **ICBYTES Library**
- **Visual Studio 2022**

Libraries and Tools:

- **ICBYTES Library:** It is a special library developed to facilitate graphical operations in the C++ programming language. It has a user-friendly interface and focuses especially on 2D graphics operations, drawing algorithms, mouse events, image processing, multi-threading support and fast data processing[1].
- **Windows API:** Windows API is used for window management, menus, file operations, threads and bitmap operations.
- **Multi-threading:** Implemented to improve drawing performance by distributing the workload.

Drawing Algorithms and Optimizations

Temporary Canvas and Drawing Management:

The project is based on three separate canvas structures:

1. Main Canvas (m): The main structure where permanent drawings are kept and freehand drawing is done.

2. Temporary Canvas (tempCanvas): Used for temporary drawings and momentary user interactions.

3. Buffer Canvas (backBuffer): The structure where all drawings are combined and projected on the screen.

During drawing operations, shapes are drawn on the temporary canvas (tempCanvas) and displayed in real time. When the user releases the left mouse button, the drawings are copied to the main canvas (m) and the tempCanvas is cleared. This structure optimizes the drawing experience by providing instant feedback to the user.

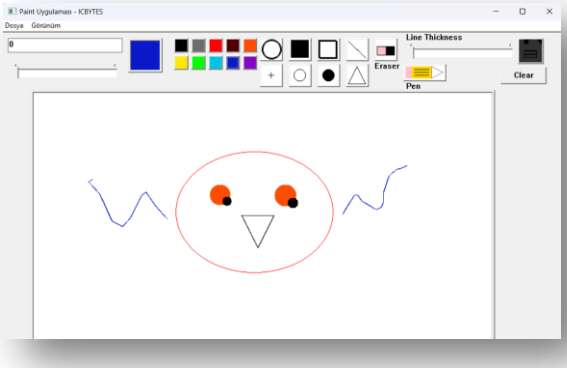

Figure 1. Paint app first look

Line Drawing Algorithm:

Using Bresenham's Algorithm, a line drawing method that minimizes the gaps between pixels is implemented. Bresenham's Line Algorithm is a fast and efficient raster line drawing algorithm. This algorithm determines pixels on a linear line using only integer operations instead of floating point calculations. It offers an approach that minimizes the gaps between pixels[2].

- ❖ With the multi-threaded implementation, the line drawing is computed in parallel.
- ❖ Thick line support is enhanced with normal vector calculations.

```
// Çizgi Çizme Fonksiyonu
void DrawLine(ICBYTES& canvas, int x1, int y1, int x2, int y2, int color) {
    int dx = abs(x2 - x1);
    int dy = abs(y2 - y1);
    int sx = (x1 < x2) ? 1 : -1;
    int sy = (y1 < y2) ? 1 : -1;
    int err = dx - dy;

    float length = sqrt(dx * dx + dy * dy);
    float normX = (length == 0) ? 0 : -dy / length;
    float normY = (length == 0) ? 0 : dx / length;

    int halfThickness = max(1, (lineThickness - 1) / 2);

    while (true) {
        // **İlk önce sadece ana çizgiyi çizer**
        Line(canvas, x1, y1, x2, y2, color);

        // **Daha sonra genişlik ekler (ilk noktada direkt kalınlaşmayı önler)**
        if (lineThickness > 1) {
            for (float i = 1; i <= halfThickness; i += 0.5) {
                int offsetX = round(i * normX);
                int offsetY = round(i * normY);
                Line(canvas, x1 + offsetX, y1 + offsetY, x2 + offsetX, y2 + offsetY, color);
                Line(canvas, x1 - offsetX, y1 - offsetY, x2 - offsetX, y2 - offsetY, color);
            }
        }

        if (x1 == x2 && y1 == y2) break;

        int e2 = 2 * err;
        if (e2 > -dy) {
            err -= dy;
            x1 += sx;
        }
        if (e2 < dx) {
            err += dx;
            y1 += sy;
        }
    }
}
```

Figure 2. Usage Bresenham's Algorithm

Drawing Circle and Ellipse:

With the Midpoint Circle Algorithm, circles were created with perfect pixelation, and calculation accuracy was increased by using parametric functions for ellipse drawing.

Performance Enhancement with Parallel Processing

Drawing operations are accelerated using multi-threading. Traditional single-threaded drawing methods can cause performance issues, especially in applications that require high-resolution graphics or intensive drawing operations. In this project, each drawing process is executed in a separate thread, making more efficient use of CPU resources.

- A thread pool mechanism has been implemented to distribute the workload evenly and utilize system resources efficiently. A more stable and faster drawing performance is achieved by reducing unnecessary thread creation and deletion costs.

- By running each drawing process in a separate thread, overloading of the main thread is avoided. This method ensures uninterrupted user interface (GUI) operations and improves responsiveness.
- Mutex (mutual exclusion) mechanisms are used to prevent multiple threads from accessing the same data structure at the same time. In this way, data inconsistency and concurrent access errors are avoided and the accuracy of drawing operations is maintained.

Drawing times have been significantly improved. Compared to the single-threaded drawing method, drawing time has been significantly reduced with the multi-threaded drawing method. A smoother and faster user experience. Thanks to parallel processing, stuttering, especially when drawing large drawings, has been avoided and processing time has been shortened. System resources are used more efficiently. A high-performance drawing experience has been achieved by getting full efficiency from multi-core processors. Thanks to these optimizations, performance bottlenecks in graphics-based applications were minimized and a real-time drawing environment was created[3].

User Interface (GUI) Design

In this project, a user-friendly and intuitive interface was designed to make the drawing experience easy and accessible. The basic functions that users may need while drawing, such as color selection, shape drawing and line thickness adjustment, are arranged in a way that is practical and quickly accessible.

Figure 3. Tools interface

Color Selection Panel:

Users can draw by selecting a color from a predefined color palette. The color preview frame allows the selected color to be shown in real time. Also, custom color tones can be set via the trackbar, thus offering a wider range of colors.

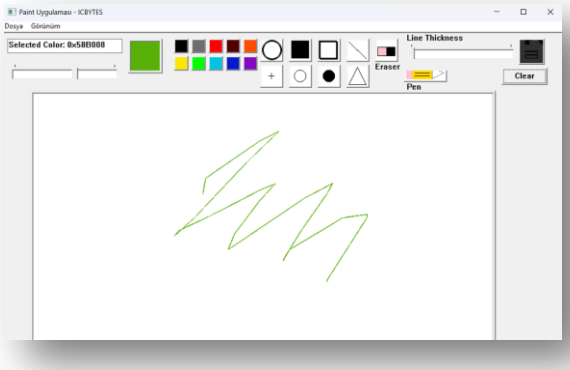

Figure 4. Color Selection

Shape Selection Buttons:

With a single click, users can draw geometric shapes such as a circle, triangle, rectangle or straight line. Each button is visualized with an icon representing the corresponding shape. By clicking on the buttons, the user can change the active drawing mode and easily add shapes.

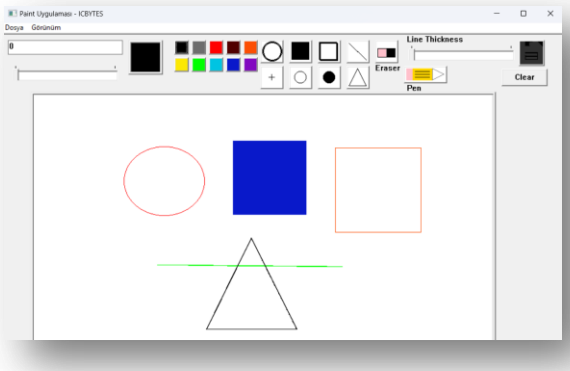

Figure 5. Shows shape selections

Line Thickness Adjustment (Trackbar):

Line thickness can be dynamically adjusted via the trackbar. The user can choose from a wide range of line thicknesses, from fine details to bold highlights. A text label just above the trackbar shows the user the current line thickness, making adjustment more intuitive.

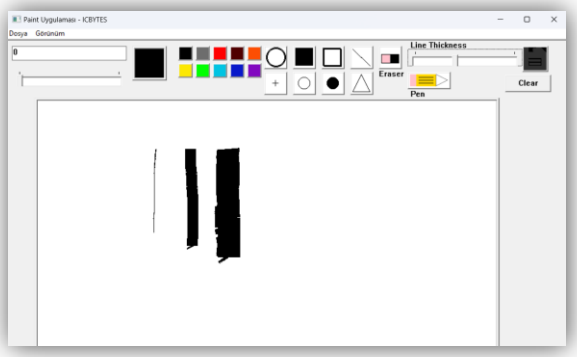

Figure 6. Shows Line Thickness

Pencil Button:

One of the main functions of the project is freehand drawing. The pen tool allows the user to draw by dragging the mouse and draws with the selected color. The line thickness can be adjusted while drawing, so different types of highlights can be made.

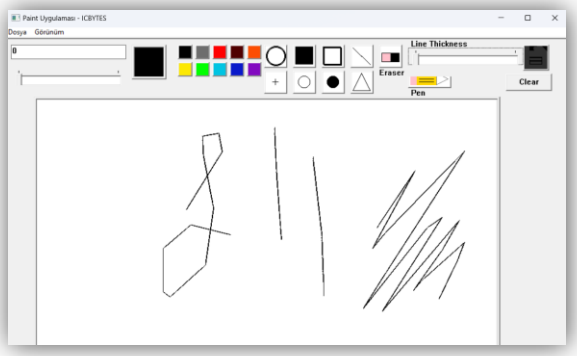

Figure 7. Drawing function

Eraser Button:

The Eraser tool, when selected, allows the user to make corrections by cleaning certain areas of the drawing area. The line thickness also affects the width of the eraser, making it possible to erase larger or smaller areas.

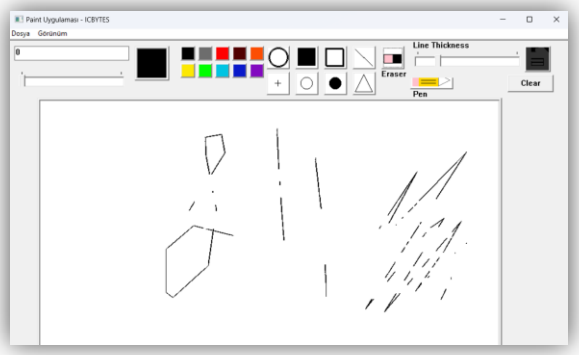

Figure 8. Showing how eraser works

Clear Button:

The Clear button has been added so that users can clear the entire canvas with a single click to reset their drawings. When this button is used, the drawing area is turned completely white and all drawings are removed and the user is given the opportunity to work on a new canvas.

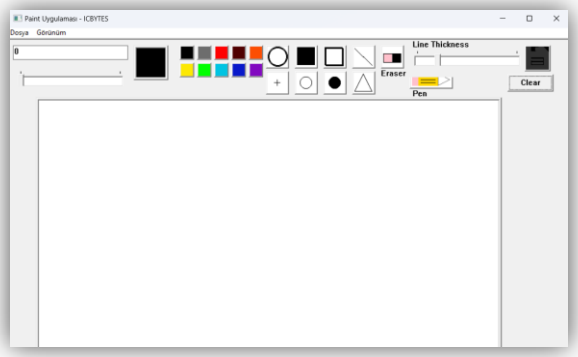

Figure 9. After clicked clear button

Save Button:

The Save button has been developed in order to save the drawings permanently. By clicking this button, the user can save the drawing as a bitmap (BMP) file. The file selection dialog box (OPENFILENAME) of the Windows API is used for the file saving process, so that the user can select the name of the file and the save location. This feature makes it possible to edit or share the made drawings later on.

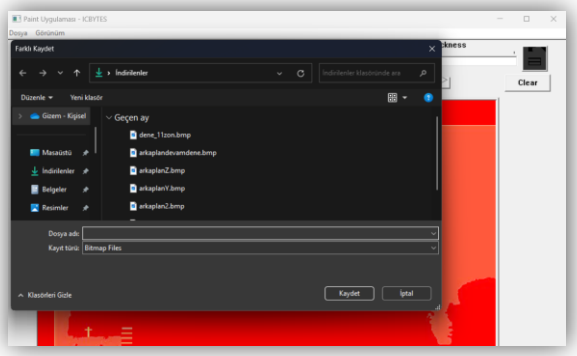

Figure 10. Save canvas

Test Process and Challenges

1. Line Thickness and Intermittent Drawing Problem: At first, a simple loop was used to increase the line thickness by painting the side pixels. However, this method resulted in uneven pixel placement in the diagonal lines, giving the lines a jagged (dashed) appearance.

Solution:

- ✓ Bresenham's algorithm was used to correctly identify the line pixels.
- ✓ Normal vector calculations were applied to create a smooth fill around the line.
- ✓ The line thickness was smoothed by adding extra lines at smaller intervals (sub-pixel rendering).

2. Performance Issue: Slow Drawing and Image Updating: Initially all drawing operations were executed in a single thread (single-threaded). This caused delays and FPS drops, especially when drawing large shapes on a high-resolution canvas.

Solution:

- ✓ A multi-threading structure was used. Each drawing process was run in parallel on a different thread.
- ✓ Thread pools were used to avoid creating unnecessary threads.
- ✓ Locking (mutex) mechanisms prevented multiple threads from changing the same data at the same time.
- ✓ Improved performance by reducing unnecessary screen updates.

3. Drawings Do Not Appear Correctly on Canvas and Coloring Problem: In the first attempts, it was noticed that geometric shapes and freehand drawings appeared in different colors. In particular, it was observed that some shapes changed color after the drawing was completed.

Solution:

- ✓ By creating a temporary canvas, the shapes were first drawn in a preview area.
- ✓ A buffer canvas was used to combine the background and foreground. When the drawing process was complete, all data was copied to the main canvas.
- ✓ Alpha (transparency) values were checked using RGBA (4-channel color format).

As a result of the problems and solutions mentioned above, the project became a more stable, fast and accurate drawing application.

Conclusion and Future Work

This project provides a high-performance drawing application with temporary canvas handling, parallel processing optimizations and advanced drawing algorithms.

Suggestions for future work:

- ❖ Providing brush effects and texture support
- ❖ Development of an undo-redo system
- ❖ Adding buttons for drawing more shapes
- ❖ Adding a box to write text on the canvas

References

[1] Baykal, İ. C. (2024). *ICBYTES: A Simplified C++ Library*. Retrieved from https://github.com/cembaykal/ICBYTES

[2] Foley, J., van Dam, A., Feiner, S., & Hughes, J. (1996). *Computer Graphics: Principles and Practice*.

[3] Microsoft Documentation: Multi-Threading in C++ (2022).