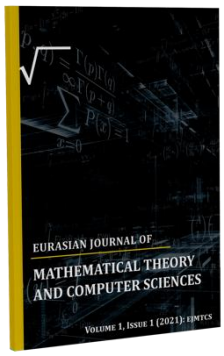
EURASIAN JOURNAL OF MATHEMATICAL THEORY AND COMPUTER SCIENCES
Innovative Academy Research Support Center
IF = 7.906 www.in-academy.uz



GRAPHICAL PROGRAMMING OF 2D APPLICATIONS IN C#

Shoyqulov Shodmonkul Qudratovich
Acting Associate Professor, department of Applied Mathematics,
Karshi State university, Karshi, Republic of Uzbekistan
https://doi.org/10.5281/zenodo.13892766

ARTICLE INFO

Received: 01st October 2024
Accepted: 04th October 2024
Online: 05th October 2024

KEYWORDS

Graphic programming, C#, WPF, GDI+, 2D graphics, animation, optimization of graphic applications, game engine, performance, interactive applications, DirectX, OpenGL, cross-platform, visualization, virtual reality, computer graphics, memory management, comparative analysis of programming languages.

ABSTRACT

This article discusses the main frameworks for C# graphical programming, such as WPF and GDI+ for working with 2D graphics and user interfaces. Examples of using each tool are given and their capabilities and advantages are analyzed. Particular attention is paid to methods for optimizing C# graphical applications to improve performance and efficient use of resources. A comparative analysis of C# with other programming languages, including C++, Java and Python, is conducted from the point of view of graphical programming. Key aspects such as performance, ease of development and application in various fields are discussed. In conclusion, it is concluded that C# is a universal and powerful tool for developing graphical applications, capable of satisfying the requirements of both novice and experienced developers.

INTRODUCTION

Graphical programming is an important and integral part of software development, which covers the creation of visual interfaces, 2D and 3D graphics, animations and interactive applications. In the modern world, graphic technologies play a key role in such areas as the creation of video games, multimedia applications, animated films, computer visualization systems and virtual reality. Effective work with graphics requires not only an understanding of the mathematical and algorithmic foundations, but also the use of modern programming languages and frameworks that support work with 2D and 3D graphics at a high level of performance [1].

The C# programming language has become one of the leading tools for developing graphic applications, due to its high performance, flexibility and powerful support from development tools such as Windows Presentation Foundation (WPF), Graphics Device Interface Plus (GDI+) and the Unity game engine. These technologies provide developers with everything they need to create complex and interactive graphic applications, while providing an intuitive syntax and powerful integration with the .NET platform. WPF, being one of the key technologies of Microsoft, allows to create high-quality interfaces for desktop applications with rich graphics, support of animations, multimedia and 2D/3D elements. Using XAML

(markup language) in WPF gives developers the opportunity to separate visual components from application logic, which simplifies the process of development and maintenance of code. In addition, WPF supports hardware acceleration, which significantly increases performance when working with graphics, making it smoother and more interactive[2]. Another important tool for graphical programming in C# is GDI+, which provides a set of APIs for working with two-dimensional graphics, such as lines, curves, rectangles and text. GDI+ was created as an extension of the classic GDI and is used to create simpler and easier to implement graphical elements, such as user interfaces and 2D graphics in desktop applications.

An important aspect of using C# for graphical programming is its ability to scalability and optimization. With the built-in language memory management tools (garbage collector), developing graphical applications in C# becomes safer and more efficient, eliminating many of the low-level issues that arise when using languages such as C++. At the same time, for complex graphical applications, it is important to understand the optimization techniques that help improve rendering performance, reduce CPU and GPU load, and use memory resources efficiently. The purpose of this article is to comprehensively consider the capabilities of the C# language in the context of graphical programming. The article will analyze the main frameworks such as WPF, GDI+ used for working with 2D graphics, and discuss optimization techniques for graphical applications. In addition, a comparative analysis of C# with other programming languages such as C++, Java, and Python will be conducted in terms of their use in graphical programming, their advantages and disadvantages in various development scenarios.

RESULTS and DISCUSSIONS

Graphics programming relies on several key concepts and theoretical foundations that define how images, shapes, and objects are represented and manipulated in a computer. These principles include working with raster and vector graphics, using two-dimensional and three-dimensional coordinate systems, and understanding mathematical operations such as transformations, scaling, and rendering.

Raster and vector graphics are the two main types of graphical representation. Raster graphics are formed using a grid of pixels, where each pixel is assigned a specific color. In C#, raster graphics can be implemented using libraries such as GDI+, where each pixel can be manipulated individually. Vector graphics are based on mathematical formulas that describe lines, curves, and polygons. In C#, vector objects can be created using WPF, which supports the construction and manipulation of graphics primitives through XAML or C# code[4].

Two-dimensional (2D) graphics are an important category in graphics programming. The problem is easily solved using technologies such as GDI+ and WPF in C#, which offer a wide range of tools for creating and manipulating 2D objects.

Transformations of graphical objects are an important component in both 2D graphics. Transformations are operations that change the position, size, or orientation of objects on the screen[6]. The main types of transformations include:

- Translation.
- Scaling.
- Rotation.

In C#, transformations are implemented using built-in methods and classes, for example, in WPF through the RenderTransform properties for 2D graphics.

Graphics programming in C# relies on a number of fundamental theoretical concepts that allow you to effectively manipulate raster and vector images, use 2D and 3D coordinate systems, and perform transformations and rendering of objects. Understanding these fundamentals is the key to successful development of graphical applications. GDI+ (Graphics Device Interface Plus) is an extension of the traditional GDI that provides developers with the ability to work with two-dimensional graphics in C# applications. It allows you to create and manipulate graphical primitives such as lines, rectangles, circles, ellipses, and also manage images and text. GDI+ is an integral part of Windows and allows developers to easily integrate graphical elements into their applications.

One of the key benefits of using GDI+ in C# is access to the Graphics class, which contains a set of methods for working with graphics. This class provides tools for drawing graphical primitives and working with images. An example of using the Graphics class is creating an application window with graphical elements such as shapes or text.

In order to use GDI+ in a C# application, you need to handle the window redraw event, which is called when the window is resized or its contents are updated. This is usually implemented using the Paint event, in which an object of the Graphics class is used to draw on the window surface.

Let's look at an example of a simple C# application that draws circles and rectangles:
using System;
using System.Drawing;
using System.Windows.Forms;
public class SimpleForm : Form { public SimpleForm() { this.Text = " GDI+ sample";
this.Size = new Size(400, 300);
} protected override void OnPaint(PaintEventArgs e) { Graphics g = e.Graphics;
Pen pen = new Pen(Color.Blue, 3);
g.DrawRectangle(pen, 50, 50, 100, 100);
g.DrawEllipse(pen, 200, 50, 100, 100);
Brush brush = new SolidBrush(Color.Green);
g.FillRectangle(brush, 50, 200, 100, 50);
}
[STAThread]
public static void Main()
{
Application.Run(new SimpleForm());
}
}

In this example, we create an application window using WinForms, and the OnPaint method handles the redraw event, in which the Graphics object is used to draw a rectangle, circle and filled rectangle. In this code, Pen controls the color and thickness of the lines, and Brush is used to fill the shapes.

One of the benefits of GDI+ is the ability to work with different styles and fonts. For example, using the DrawString and MeasureString methods, you can easily display text on the screen with with the specified font and color parameters:

g.DrawString("Hello, world!", new Font("Arial", 16), Brushes.Red, new PointF(50, 150));

This code displays text on the screen using the Arial font 16 pixels in size and red in color.

Optimizing work with GDI+ is an important aspect when developing graphics applications. Because each drawing operation requires resources, it is important to avoid unnecessary redraws. For example, you can optimize by using double buffering techniques, which prevent the screen from flickering when graphics are updated. In C#, you can enable double buffering by setting the DoubleBuffered property on a form or panel:

this. DoubleBuffered = true;

Using GDI+ provides flexible and powerful tools for creating graphical interfaces in C# applications. However, when working with more complex graphics or high performance requirements, such as animations and complex visual effects, developers may encounter the limitations of GDI+. In such cases, it is worth considering other technologies, such as WPF or DirectX.

GDI+ remains a useful tool for developing applications with 2D graphics, offering a convenient and understandable interface for working with graphical elements. It is especially effective for simple tasks, such as drawing primitive objects and displaying text information.

Windows Presentation Foundation (WPF) is a powerful framework developed by Microsoft for creating modern graphical user interfaces (GUIs) in C# applications. WPF provides more flexibility and advanced capabilities than traditional approaches such as WinForms or GDI+. One One of the key advantages of WPF is its ability to work with both two-dimensional (2D) and three-dimensional (3D) graphics, making it a versatile tool for creating visually rich and interactive applications. An important feature of WPF is the use of the declarative language XAML (Extensible Application Markup Language) to define interfaces. XAML allows you to separate the visual part of the application and the logic of its operation, which simplifies development and makes the code more structured and readable. At the same time, programmers can write interfaces both directly in XAML and through C# code.

Creating graphical interfaces using WPF begins with working with a XAML file, which defines the structure of the window and its visual elements. Let's look at an example of a simple interface that contains buttons, text boxes, and a rectangle:

```
<Window x:Class="WPFExample.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title=" WPF Sample" Height="350" Width="525">
<Grid>
<!-- Bar -->
<Rectangle Fill="Blue" Width="200" Height="100" VerticalAlignment="Top" Margin="10"/>
<!-- TextBox -->
```

<TextBox Width="200" Height="30" VerticalAlignment="Center" HorizontalAlignment="Left" Margin="10"/>

<!-- Button -->

<Button Content="Click Me" Width="100" Height="30" VerticalAlignment="Bottom" HorizontalAlignment="Left" Margin="10"/>

</Grid>

</Window>

This example creates a simple window containing three elements: a rectangle, a text field, and a button. Each element is defined via XAML, and developers can easily change its appearance and behavior by specifying parameters such as color, size, location, and other properties. This declarative nature greatly simplifies the creation of complex interfaces, since all elements can be flexibly configured via markup without resorting to complex program logic.

One of the key advantages of WPF is its advanced graphics support, including vector graphics and multimedia. WPF uses hardware acceleration via DirectX, which allows you to create interfaces with smooth animations and complex graphical effects without significant performance costs.

- Vector graphics: Unlike raster graphics, vector graphics in WPF allow you to scale elements without losing quality. This is especially useful for interfaces that require flexibility when changing screen resolution or window sizes. Vector elements such as lines, polygons, and curves are easily created through XAML using objects such as Line, Polygon, Ellipse, and others.
- Animation. WPF supports complex animations of objects such as movement, rotation, resizing, and transparency. This can be used to create more attractive and intuitive interfaces. An example of an animation that specifies a change in the transparency of an element can be implemented as follows:

<Rectangle Width="200" Height="100" Fill="Red">

<Rectangle.Triggers>

<EventTrigger RoutedEvent="MouseEnter">

<BeginStoryboard>

<Storyboard>

<DoubleAnimation Storyboard.TargetProperty="Opacity" From="1.0" To="0.5" Duration="0:0:1"/>

</Storyboard>

</BeginStoryboard>

</EventTrigger>

</Rectangle.Triggers>

</Rectangle>

In this example, the animation decreases the transparency of the rectangle when the mouse cursor hovers over it.

One of the most important aspects of WPF is the ability to handle events. User events, such as button clicks or text input, are handled using event methods in C#. For example, to handle a button click, you can use the following code:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
MessageBox.Show("Button is pushed!");
}
```

Binding the logic to XAML is done using the Click attribute, which points to the handler method:

```
<Button Content="Click button" Click="Button_Click" Width="100" Height="30"/>
```

Using WPF to create graphical interfaces greatly simplifies the development process thanks to powerful tools for working with graphics, multimedia, and animation. WPF provides a flexible and intuitive way to create modern interfaces where you can easily combine 2D and 3D graphics, interactive elements, and complex animations. Hardware acceleration through DirectX allows you to create visually rich applications with high performance. Optimizing graphics applications is a key aspect of development, especially in cases where performance is critical. Graphics applications, whether games, visual simulations, or complex graphics editors, require high performance to process large amounts of data, smooth animation, and real-time interactivity. C# applications allow you to use various optimization techniques to improve performance, reduce CPU and GPU load, and minimize rendering delays. Buffering is another important optimization technique. It involves storing graphic content in memory, which avoids re-rendering unchanged elements. Instead of recalculating and drawing objects each time, an application can save an image in a buffer and display it on the screen without unnecessary calculations. Double buffering is a technique used to prevent visual artifacts such as flickering or tearing. In this approach, the image is first drawn to a hidden buffer in memory and then displayed on the screen. This helps ensure smooth and consistent graphics[12].

Graphics applications can benefit significantly from hardware acceleration, which allows the graphics card (GPU) to handle graphics processing. Instead of relying solely on the CPU, rendering operations are offloaded to the GPU, which is better able to handle the parallel computing required to handle graphics.

Memory management is an important part of optimizing graphics applications. In C#, the CLR automatically frees unused objects through garbage collection. However, excessive creation and deletion of objects, such as textures or large data structures, can cause sudden delays in application execution, as the garbage collector may temporarily stop program execution to free up memory[3].

Optimizing graphics applications in C# requires careful resource management and graphics processing. Key techniques include minimizing render calls, using buffering, hardware acceleration, and proper texture management. Performance profiling and analysis help to find bottlenecks and improve the efficiency of the application. These approaches allow to create graphical applications with high performance and smooth visualization, which is especially important in the development of games and interactive applications[5].

Graphical programming requires high performance, flexibility and a wide range of tools for working with 2D and 3D graphics. The C# language is widely used in this area, especially due to its tight integration with frameworks such as WPF, GDI+. However, there are many other programming languages on the market, such as C++, Java, and Python, which are also

actively used for developing graphical applications. A comparative analysis of C# with these languages allows us to evaluate their advantages and disadvantages in the context of graphical programming.

CONCLUSIONS

Graphics programming in C# provides developers with powerful tools for creating visually rich and interactive applications, especially thanks to technologies such as WPF, GDI+, and Unity. C# allows you to develop both simple 2D interfaces and complex 3D scenes, providing high performance and flexibility. The main advantage of C# is its simplicity, intuitiveness, and integration with modern graphics engines, making it an excellent choice for both beginners and experienced developers.

Compared to other programming languages such as C++, Java, and Python, C# strikes a balance between ease of development and the ability to create high-performance applications. Although C++ still remains the leader in the development of graphics engines and applications with high resource requirements, C# offers a more convenient environment for rapid prototyping and cross-platform application development through Unity. At the same time, compared to Java and Python, C# offers greater opportunities for creating 3D graphics and games using hardware acceleration.

Optimizing graphics applications in C# includes key techniques such as minimizing rendering calls, using hardware acceleration, and optimizing memory handling. These methods allow you to create applications that run faster and more efficiently, which is especially important when developing games and interactive simulations.

In the future, as graphics technologies develop and computing power increases, C# and its ecosystem will continue to evolve and remain one of the leading tools for graphics programming, providing developers with even more opportunities to create innovative and high-performance graphics solutions.

References:

1. Shoyqulov Sh. Q.METHODS FOR PLOTTING FUNCTION GRAPHS IN COMPUTERS USING BACKEND AND FRONTEND INTERNET TECHNOLOGIES. European Scholar Journal (ESJ). Vol. 2 No. 6, June 2021, ISSN: 2660-5562. P.161-165, https://scholarzest.com/index.php/esj/article/view/964/826
2. Shoyqulov Sh. Q., Bozorov A. A. Methods for graphing functions in computers using Web technologies. Journal of Information Computational Science. Journal Vol. 1 Issue 1, JUNE 2021. Urgench., https://www.sciencepublish.org/index.php/ics/article/view/79
3. Shoyqulov Sh. Q. Wonderful multimedia - applying in areas outside of teaching. “Innovations in technology and science education” scientific journal, Volume #2, issue#7, Publication: february 2023, p. 700-708, SJIF-5.305, ISSN 2181-371X, https://humoscience.com/index.php/itse/index
4. Sh.Q. Shoyqulov. (2021). Methods for plotting function graphs in computers using backend and frontend internet technologies. European Scholar Journal, 2(6), 161-165. Retrieved from https://scholarzest.com/index.php/esj/article/view/964
5. Sh.Q. Shoyqulov, A. M. Shukurov. Propagation of Non-Stationary Waves Of Transverse Displacement from a Spherical Cavity in an Elastic Half-Space.

6. International Journal of Advanced Research in Science, Engineering and Technology. 13291-13299. Vol. 7, Issue 4 , April 2020. http://www.ijarset.com/upload/2020/april/13-shshovqulov-02-1.pdf
7. Shoyqulov Sh. Q., Bozorov A. A. Methods for plotting function graphs in computers using modern software and programming languages. ACADEMICIA: An International Multidisciplinary Research Journal. 321-329. 2021, Volume : 11, Issue : 6. ISSN : 2249-7137. DOI : 10.5958/2249-7137.2021.01619.0. Online published on 22 July, 2021.
8. Bozorov Abdumannon, & Shoyqulov Shodmonkul Qudratovich. (2022). MULTIMEDIA SURVEILLANCE CAMERAS AND THEIR FEATURES IN USING. Open Access Repository, 9(10), 29–34. https://doi.org/10.17605/OSF.IO/4EV75
9. Bozorov Abdumannon, Nodirbek Abdulkhayev, Shoyqulov Shodmonkul Qudratovich. (2022). MODERN TECHNOLOGIES OF VIRTUAL REALITY– A NEW MULTIMEDIA OPPORTUNITIES. EURASIAN JOURNAL OF MATHEMATICAL THEORY AND COMPUTER SCIENCES, 2(11), 85–90. https://doi.org/10.5281/zenodo.7251370
10. Qudratovich, S. S. (2022). The Role and Possibilities of Multimedia Technologies in Education. International Journal of Discoveries and Innovations in Applied Sciences, 2(3), 72–78. Retrieved from http://openaccessjournals.eu/index.php/ijdias/article/view/1148
11. Qudratovich, S. S. (2022). Technical and Software Capabilities of a Computer for Working with Multimedia Resources. International Journal of Discoveries and Innovations in Applied Sciences, 2(3), 64–71. Retrieved from http://openaccessjournals.eu/index.php/ijdias/article/view/1147
12. Sh.Q. Shoyqulov. (2022). The text is of the main components of multimedia technologies. Academicia Globe: Inderscience Research, 3(04), 573–580. https://doi.org/10.17605/OSF.IO/VBY8Z
13. Sh.Q. Shoyqulov. EditorJournals and Conferences. (2022, May 3). The graphics- is of the main components of multimedia technologies. https://doi.org/10.17605/OSF.IO/2KAM8
14. https://wos.academiascience.org/index.php/wos/article/view/1427
15. Shoyqulov, S.Q. and Bozorov, A.A. 2022. The Audio- Is of the Main Components of Multimedia Technologies. International Journal on Integrated Education. 5, 5 (May 2022), 263-268.
16. Shoykulova Dilorom Kudratovna, & Sh.Q. Shoyqulov. (2022). PHP is one of the main tools for creating a Web page in computer science lessons. Texas Journal of Engineering and Technology, 9, 83–87. Retrieved from https://zienjournals.com/index.php/tjet/article/view/2000