Resource Aware Implementation of Image Processing Algorithms – A Teacher Perspective

Petar Rajković[1, *, †], Dragan Janković[1, †]

[1] University of Niš, Faculty of Electronic Engineering, Aleksandra Medvedeva 14, Niš, Serbia

Abstract

Teaching image processing algorithms is a widely interesting and easily accepted topic by students. The mentioned algorithms offer initial visual results, a lot of space for improvements, personal initiative, and the opportunity for continuous work. This paper presents our experience in teaching image processing algorithm implementation with different approaches, emphasizing concepts of resource awareness. The programming routines explained to the students start from a managed environment with and without dedicated classes and packages, through the unsafe code execution up to native code integration. The paper presents the comparison of time utilization, programming effort, and the level of the concept adoption by the students. The analysis was based on the student projects uploaded to the learning management system in the period from 2018 to 2024. The results are used to adjust the course structure and better adopt resource awareness-related concepts. The percentage of projects entirely based on more effective approaches rose from 44% in 2018 to 80% in 2023 and 2024. During the monitored period, the overall average execution time of the benchmark algorithms was reduced closely to one-third compared to the results from 2018, which follows the shift towards more effective approaches. In this way, we tend to say that it is important to point out all the technology benefits and shortcomings and to encourage students to try to find more effective ways to solve time and resource-consuming problems.

Keywords

Image processing algorithms, resource awareness, execution efficiency

1. Introduction

Implementation of image processing algorithms is considered a remarkably interesting part of computer science [1]. Many students and programmers genuinely like this topic since it gives results that can be verified quickly and visually.

To learn about image processing, a variety of sources are available through different books, papers, and tutorials. Looking at Google Scholar, it could be realized that standard learning resources like [2] and [3] are cited several thousand times. Furthermore, the topic offers particularly good ground for students' further development and research.

Teaching image processing itself, is also an interesting challenge. It is important to find a suitable approach in terms of the used technology, offered frameworks, and teaching methods. All this should consider previous students' knowledge and experience. The result should be a well-defined engineering course and a properly suited set of requirements for the projects that should be implemented by the students.

One of the well-known approaches, that we used as the starting point, is to create an environment with basic methods and interfaces that will allow students to implement their algorithms as plugins [4]. This methodology typically favors technologies that enable rapid implementation, such as Java or C#, with the objective being to grasp complex algorithms. In our scenario, we present students with a project that includes fundamental functionalities and an application framework, yet they retain the liberty to develop their applications from the start.

Since we teach the course in the eighth (final) semester of bachelor studies, we tend to move the focus of the subject to more efficient programming, while basing the course outline on the standard image processing algorithms that could be practically used, as

3rd workshop on Resource AWareness of Systems and Society (RAW 2024)
* Corresponding author.
† These authors contributed equally.
petar.rajkovic@elfak.ni.ac.rs (P. Rajković);
dragan.janković@elfak.mi.ac.rs(D. Janković);
0000-0003-4998-2036 (P. Rajković); 0000-0003-1198-0174 (D Janković);

in [5] and [6]. The idea of focusing on real-life effects, such as execution speed, was taken from [5], while topics from the research shown in [6] helped in forming the curriculum.

On the other hand, the image processing algorithms are interesting from the point of view of execution optimization and general resource awareness, using different techniques and approaches [7][8].

In this study, emphasis is placed on examining the students' execution of image processing algorithms and their propensity to adopt various resource-aware strategies to enhance performance in their projects. Besides the execution of complex algorithms constitutes a component of the curriculum at the bachelor's, master's, and doctoral study levels. This analysis concentrates on a cohort of final-year bachelor's degree students who undertake image processing algorithm implementation as part of their Multimedia Systems course. [9].

To implement some operations on digital images, they must be stored in the form of a matrix of pixels [1]. The operation should run through the matrix and perform some calculations for each pixel. To describe the importance of resource awareness in image processing, let us analyze the processing requirements for the simple invert operation [3]. Invert operation is replacing every byte in the picture with its complementary value. I.e., if one byte in the original picture has a value of 30, in the inverted picture it will have a value of 225. The inverted value is calculated as the difference between the maximal value for a byte and the original value.

If the image, where we perform the invert operation, has a resolution of 1000x1000 pixels, and three bytes per pixel (standard 24bRGB format [10]), this means that three million operations need to be performed to create the inverted image.

If we observe today's image resolution, which is far more than one million pixels, it is obvious that image processing must be implemented carefully and effectively. Looking at the side of the story related to the execution speed, the obvious direction is to go towards the implementation in the programming environment closest to the processor and operation system's core.

Unfortunately, such environments are not usually user-friendly and not convenient for the one-semester course teaching where students must finish their tasks and learn as much as possible. On the other hand, the environments based on the execution of virtual machines, such are Java and .NET offer high programming flexibility and user-friendly interface which makes the work on the project much faster. Unfortunately, the execution speed in such environments is much slower than with native code.

For this reason, we choose one possible approach that could bring the best of both worlds, with the technology that offers a combination of both execution speed and fast developments, in the parts where each of the approaches has its value [14].

At the same time, we do not post any limitations to the students for the technology choice, but advocate implementation approaches that result in the code that executes faster. However, in this paper, four different implementation approaches, which are presented to the students during the course, are compared by the execution and development speed, with a set of guidelines on how to use them.

The mentioned approaches are used by the students for the implementation of various categories of image-processing algorithms, as part of their projects. Data collected over several years are examined to identify the student's responses to the use of the different implementation approaches.

The study shows that students are willing to adopt implementation approaches that are more complex, from the point of view of the implementation, up to some point, for the benefit of faster code execution.

2. Course Environment and Project Implementation

It should be noted that each student that attended our course, hails from the Department of Computer Science and is presently in their final semester. Throughout their academic journey, they have uniformly completed a foundational curriculum encompassing algorithms, mathematics, as well as programming environments and technologies. Consequently, there exists no subset of students possessing a disproportionate advantage in terms of prior knowledge. Prior to commencing our course, they have garnered experience with programming languages such as C++, Java/C#, in addition to various Web and Mobile technologies.

The example project environment is set up to Microsoft .NET Windows Forms application [11]. Such setup is advocated by the fact that the technology has all the prominent features of object-oriented design (similar to Java in [4]), is constantly developed, and has a fully integrated development environment offering the possibility to build user interfaces fast and debug efficiently.

Like Java, .NET is the environment that runs on a process virtual machine [12]. In that sense, it offers higher security for the end-user in terms of garbage collection, exception handling, and managed code execution. The downside of this approach is slower execution.

```
static void UnsafeTest()
{
    int value = 2023;

    unsafe
    {
        int* pValue = &value;

        *pValue = 2024;
    }

    Console.WriteLine(value);
}
```

Figure 1: Method with an unsafe block.

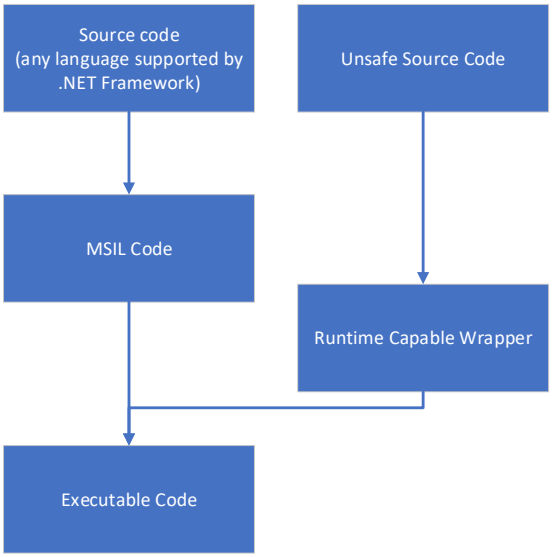

Figure 2: Integration of managed and unmanaged code (as described in [12])

To overcome this issue, the application requires a higher execution speed, the .NET framework offers the concept of unsafe code execution [14] which allows the programmer to write native C++ code into a managed environment directly (**Figure 1**). Furthermore, unsafe allows full pointer level memory access without restriction. The unsafe code is written under a specific block starting with the keyword unsafe.

Also, .NET offers the possibility to directly include libraries written in native code, as in any other programming environment, which is supposed to run as fast as possible. To support lectures, the following implementation modes are presented to the students. Namely:

- Raw data processing through managed code
- Included capabilities of framework classes (managed code)
- Processing based on the native code, in the unsafe block, which runs through the managed environment.
- Processing based purely on a native code, written, and evaluated in an external execution environment, and then brought to the managed environment using COM (Component Object Model) interface and runtime-capable wrapper.

As presented in **Figure 2**, the source code, crafted in .NET languages such as C#, is initially translated into intermediate or bytecode. Subsequently, this bytecode is compiled into the final executable. In contrast, unsafe code bypasses the intermediate code stage and is compiled directly into the executable. Additionally, integrated calls to native libraries are executed directly in the executable code, leveraging the capabilities provided by the COM interface.

2.1. Course and Project Outline

The curriculum is delivered over the spring semester, spanning 14 working weeks, and encompasses a variety of topics related to image processing. In the initial stages, students become acquainted with various color models, image file formats, and techniques for reducing image size, including downscaling and compression methods.

Subsequently, the course curriculum introduces students to a range of image processing algorithms, including fundamental filters (such as brightness, grayscale, contrast, gamma correction, and others), dithering techniques, convolution, and displacement filters. Furthermore, advanced subjects are explored, encompassing histogram-based methods and intricate filters like the Kuwahara filter. [13].

When teaching image processing algorithms we focus on three major points – algorithm construction/correctness, variants and similar filters, and execution efficiency. In most cases, execution efficiency is more important than memory use, since many algorithms are built around additional data structures which cannot be avoided during the implementation.

As mentioned, the focus of our course is execution analysis. Since we expect that students implement algorithms efficiently, from the algorithm flow's point of view, we display the differences in the execution depending on the chosen technology.

The students are then able to measure in the demo filters differences in the execution speed between the implementation in different technologies and directly experience trade-offs between comfort programming environments (such as C# and Java), and execution speed in native code implementation (C++).

Since the execution time is one of the most important factors in the eventual grade (14 out of 30 points), students could decide whether pays off if they invest more time in a faster solution (Table 1).

Over the years, we slightly adapted the project structure, taking into consideration results from the previous years. The current project structure consists of:

- Image processing application that can load standard file formats and display changes after applying image transformation.
- Defining own file format by implementing down sampling and compression
- Implementation of one basic filter
- Implementation of one dithering method (like convolution filters)
- Implementation of one displacement filter
- Additional advanced filter implementation

During this time, we kept the project complexity requirements at the same level. The basic requirement is to implement a user-friendly application that will display, at any point, original and changed pictures, and to allow the user to save the effect of transformation either to some standard or to a custom file format.

From year to year, we are changing requirements in terms of classes of implemented algorithms as well as the definition of the file format. The projects from one year exclude combinations of algorithms implemented in previous years, but the general requirements remain at the same level. For example, this year (2024) the students had to convert RGB pictures to YUV models and then apply some downsample scheme as in JPEG (4:2:0, or 4:1:1, etc.). After downsample, they had to implement some of the dictionary based lossless compression algorithms like Huffman or Shannon-Fano. Table 2 shows the requirements for filters in three different years.

This year students had to implement contrast as the basic filter, where different variants will be applied to different group of students. Similarly, for the convolution style filter, dithering was the theme for 2024, where each group should implement its specific variant (like Jarvis, Stucki, Sierra, etc.).

When we look in the Table 1, each project element comes in two to five variants, giving us more combinations than the students, which means that every student will have a unique project. I.e., in 2024, we had 4 downsampling variants, 2 compression algorithms, 3 contrast variants, 4 dithering filters, 3 variants of displacement filter, and 2 different advanced topics, which makes 576 different variants for less than 100 students.

Table 1
General point distribution scheme and criteria

Project element	Maximal points	Correctness	Execution time
Application structure	2	2	n/a
Downsampling	4	2	2
Compression	4	1	3
Basic filter	2	1	1
Convolution filter	5	2	3
Displacement filter	5	3	2
Advanced topic	6	3	3
Code quality	2	2	n/a
Total	30	16	14

Table 2
Examples of project composition between different years (Project elements: B – basic filter, C – convolution-style filter, D – displacement filter, A – advanced topic)

Project element	2018	2021	2024
B	Gamma	Grayscale	Contrast
C	Edge detect	Sharpen	Dithering
D	Pixelate	Time-Warp	Water
A	Trans-domain	Histogram	Kuwahara

The students upload their projects in Moodle (until 2020) and Teams (2021 onwards). After that, we execute them in the referent environment and with a referent set of images and assign points for each project item. We evaluate both algorithm correctness and execution speed. In all these years evaluation has been done only by the authors of these papers, which, we believe, ensured the same level of assessment. In this work, we are focused only on execution time. Since the student applications execute a single algorithm at a time, memory usage and scalability are out of the scope of their projects. In the same semester, they have different course (Distributed systems) where the main aim is on scalability and robustness).

2.2. Raw Data Processing

Raw data processing is the basic way to process images (**Figure 3**). The programmer should load an array of bytes from the file and transform them into a meaningful data structure. The complexity of this task depends on the file type. For example, uncompressed files like bitmaps could be directly loaded and converted, while compressed files, like JPEGs and PNGs, must be significantly processed.

```
84 public static bool InvertDirect(string sourcefileP
85                                   string destFilePat
86 {
87     FileStream fsr = File.OpenRead(sourcefilePath)
88     byte[] a = new byte[fsr.Length];
89     fsr.Read(a, 0, Convert.ToInt32(fsr.Length));
90
91     FileStream b = File.OpenWrite(destFilePath);
92
93     int pos = a[10] +
94         256 * (a[11] +
95                 256 * (a[12] +
96                         256 * a[13]));
97
98     for (int i = 0; i < fsr.Length; i++)
99     {
00         if (i < pos)
01         {
02             b.WriteByte(a[i]);
03         }
04         else
05         {
06             byte res = (byte)(byte.MaxValue - a[i]
07             b.WriteByte(res);
08         }
09     }
10
11     b.Close();
12     return true;
13 }
```

Figure 3: Method invert written as raw data processing.

For bitmaps [15], every byte has its meaning, and the corresponding piece of information could be extracted. I.e., in the presented example, 4 bytes started at position 10 in the file defining image height and the next 4 image weight. The bytes starting from position 54, for images stored as 24b RGB files, define the image, and knowing that each pixel is described by three bytes makes the image processing straightforward.

The implementation based on raw data processing is suitable only for plain bitmap files since they contain directly stored pixel-related data. This approach requires no additional data structures and classes and could be implemented using the programming language basics only. In this sense, it requires a bit more organization of the processing and it could be suitable for the filters with simpler implementation. This approach could be used in any programming language, and if properly implemented, could give excellent execution performance.

2.3. Processing Based on Framework Classes

For the most effective use, the framework classes could be utilized. The most important class for image processing is the class Bitmap inherited from the more common class Image [16].

This class offers the easiest, from the programming point of view, approach. The programmer loads a picture of any supported type using the Load method. Once, the image is loaded and converted to a bitmap object, each pixel can be accessed by *GetPixel* method.

```
33 public static bool Invert(Bitmap bitmap)
34 {
35     for (int i = 0; i < bitmap.Width; i++)
36     {
37         for (int j = 0; j < bitmap.Height; j++)
38         {
39             Color c = bitmap.GetPixel(i,j);
40             Color newColor = Color.FromArgb(
41                             byte.MaxValue - c.R,
42                             byte.MaxValue - c.G,
43                             byte.MaxValue - c.B);
44             bitmap.SetPixel(i, j, newColor);
45         }
46     }
47
48     return true;
49 }
```

Figure 4: Method invert written in managed code with supporting Bitmap class and Color structure.

This method returns an instance of type Color with Red, Blue, Green, and Alpha components. These values could be then processed, and new color values now could be written back to the Bitmap object. From the logical point of view, the programmer must focus only on the implementation of processing algorithms, without the need to take care or pay attention to any other task in the scope of image processing.

On the other hand, this approach runs fully under managed code, and it is the slowest way of implementation. One can say that the programming comfort is paid by the slowest runtime.

Development using the provided framework classes [16] is the easiest from the programmer's point of view (**Figure 4**). With the simple call of the single method, the programmer will have a completely structured image converted to a bitmap object with all the features and properties directly exposed. The downside of this approach is that the code execution is the slowest. The execution time is directly proportional to the size of the image, and additional time will be spent on locking and unlocking the memory area for every single execution of *SetPixel* function.

The calls of the methods that should get or set the pixel value must go through the execution virtual machine and must ensure necessary locking mechanisms each time. The recommendation for this approach is to be used in the initial stage of the projects where the students are setting up their environment and learning the process of image processing itself. This approach could be used if the expected size of the picture will not exceed certain limits, and if it will be used for previews since they could be easily integrated into Web routines and technologies such as Blazor.

2.4. The Use of Unsafe/Native Code

There are two options to use unsafe/native/non-managed code. First, the programmer could write a method in the native language library and then include the library in the project (**Figure 5**). The problem with this approach is that debugging of such code should be done in some programming tool, different from the environment where the main application is written.

```
[System.Runtime.InteropServices.DllImport("user32.dll")]
1 reference
private static extern int MessageBox(IntPtr hWnd,
                                     string text,
                                     string caption,
                                     uint type);
```

Figure 5: Declaration of the external function written in native code.

It is important to point out that such an approach is not a problem per se during the development of real software systems, but for student projects whose duration is very limited, and where students have to cope concurrently with the additional tasks, learning to use multiple environments could be a challenge up to some point.

To have the best of both worlds – managed and unmanaged code, the students are encouraged to use unsafe blocks in the Windows Forms application [18]. This approach is not common for other development environments, such is Java, and it is considered more like an additional than the technology standard.

Working in the unsafe block is close, if not equal to writing code in the native, C++ environment. The programmers have under their disposal, complete pointer arithmetic with the additional requirements to transform data types from .NET to native classes, and to take care of garbage collection.

The implementation of the invert operation in an unsafe environment is displayed in **Figure 6**. To support the coding, programmers could use *BitmapData* [17] class and its properties. Conversion from Bitmap to *BitmapData* is done using *LockBits* and *UnlockBits* methods, which ensure, in addition, uninterrupted memory management. The area of memory that stores raw image bytes will be safely locked before the processing moves to an unsafe environment and then unlocked when the program flow returns.

```
public static bool InvertUnsafe(Bitmap bitmap)
{
    BitmapData bitmapData =
        bitmap.LockBits(new Rectangle(0,
                                      0,
                                      bitmap.Width,
                                      bitmap.Height),
                        ImageLockMode.ReadWrite,
                        PixelFormat.Format24bppRgb);

    IntPtr Scan0 = bitmapData.Scan0;

    unsafe
    {
        byte* pixelPointer = (byte *)(void *)Scan0;
        int widthInBytes = bitmap.Width * 3;

        for (int i = 0; i < bitmap.Height; i++)
        {
            for (int j = 0; j < widthInBytes; j++)
            {
                pixelPointer[0] =
                    (byte)(255 - pixelPointer[0]);
                pixelPointer++;
            }
        }
    }

    bitmap.UnlockBits(bitmapData);

    return true;
}
```

Figure 6: Method invert written in unsafe code.

The approach with an unsafe code execution is the fastest way to perform image processing from the managed code environment. It is up to some percent slower combined with direct native applications, but the comfort of the integrated development seems justified as the acceptable price.

As can be seen in the previous section .NET environment offers adequate transformation classes that help in moving the execution context from managed to unmanaged. Keeping in mind that the application development, in the case of the student project, relates to tight deadlines such an approach proved its value.

2.5. Brief on Implementation Approaches

As it could be seen each of the approaches has its benefits and drawbacks. While raw data processing requires no additional libraries and frameworks, it requires more attention to organize code, and it is not useful for complex image types.

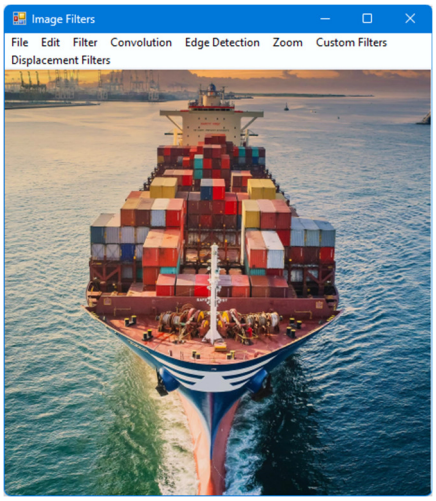

Figure 7: An example of the demo picture used for filter evaluation, taken from the Bing wallpaper site https://bing.gifposter.com/au/column-41-container-ship-near-a-commercial-port-in-thailand.html, and then cropped to 2000x2000

Relying only on the framework based on managed code will speed up the development process, at the price of the slowest execution. The integration of externally developed algorithms brought the fastest execution, but the development must be split between multiple projects, development tools, and programming languages. It requires the highest amount of time for development.

The approach based on the unsafe code seems like a promising approach for student projects focused on image processing algorithms. It offers performance close to the native code, seamless integration of the native code in the managed environment, and higher memory efficiency since it enables precise memory management. The drawbacks of this approach are the same as with the native code execution - higher potential for bugs and higher complexity for maintenance [18]. The images used for the evaluation are in resolution of 2000x2000 pixels (**Figure 7**) and higher with different aspect ratios and color representation.

3. Classes of Image Filters for Demo

For the application of the mentioned approaches, the students have the task of implementing several image filters from the various categories. To make the most convenient test cases, three diverse types of filters are checked:

- Basic filters – their implementation requires only iteration through all pixels in the image. The only programmer required to translate managed to native code.
- Convolution filters – the filters where the values of neighboring pixels affect the currently processed pixel. Compared to basic filters they require significantly more processing, but a similar amount of memory
- Displacement filters – require additional data structure which needs a comparable amount of memory as the processed image. Compared to basic filters they require more memory and more processing operations

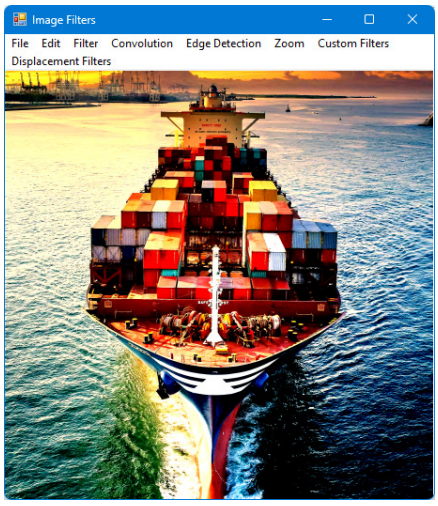

Figure 8: Example of applied contrast filter.

3.1. Basic Filters

As has been mentioned, basic filters are those that require one pass through the matrix of pixels with an optional parameter transformation. For our evaluation, we choose a contrast filter (**Figure 8**), which is an example of a common filter with many specific variants depending on the area of the application [19]. Aside from this one, the students had a few more as part of the task - brightness, color, gamma, grayscale, and conversion to distinct color models [20] – like luma-chroma (or YUV) or hue-saturation-value (HSV).

3.2. Convolution Filters

Unlike the basic filters, the convolution filters use the auxiliary matrix structure, usually called the kernel. The kernel is a matrix of small size, in most cases 3x3, and is set up with predefined parameters. In addition, the convolution filter has two more values - factor and offset. Some of the example of convolution matrices are presented in **Figure 9**.

Sharpen
- Offset: 0
- Factor: n - 8

0	-2	0
-2	n	-2
0	-2	0

Emboss Laplacian
- Offset: 127
- Factor: 1

-1	0	-1
0	4	0
-1	0	-1

Edge detect quick
- Offset: 0
- Factor: 1

-1	-1	-1
0	0	0
1	1	1

Figure 9: The examples of convolution filters with corresponding kernel matrices, offset, and factor values.

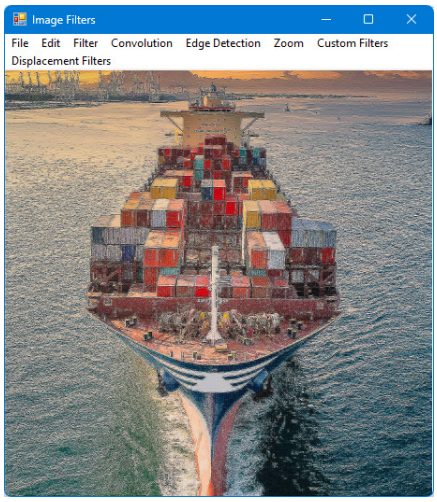

Figure 10: Example of the applied convolution filter

The processing works in a way that the submatrix, of the same size as a kernel, from the picture, should be extracted. The value of each pixel from the extracted matrix is multiplied by the corresponding parameter in the kernel. All these products are then summed up together, divided by the value provided for factor, and on top of this value is added offset.

The values in the matrix are important for the filtering operation itself, while the values for factor and offset are used to normalize the sum of products into the required value range. In our case, these are the values that could be stored in one byte (0 to 255).

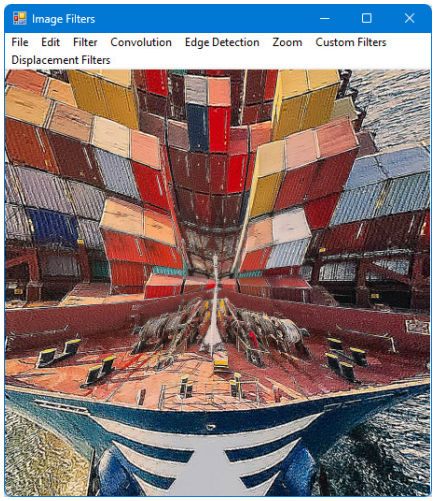

Figure 11: Example of displacement time warp filter.

The number of operations needed to perform the convolution filter is significantly higher than with basic filters. We can assume that during basic filter execution, the number of performed steps is the multiplication of the image resolution, for the convolution filters it is a multiplication of the image resolution and number of elementary operations connected with a chosen kernel matrix. For example, the execution of a sharpened filter (**Figure 10**) will execute ten times (nine multiplication and division with the factor value) more operations than the execution of a basic invert.

3.3. Displacement Filters

Displacement filters are based on a pre-built transformation matrix which is the same size as the targeting picture (**Figure 11**). Their execution is split into setup and execution phases. In the setup phase, the transformation matrix is created using some algorithm, and in the execution phase, the values from the transformation matrix are combined with the pixel values from the picture to achieve the desired effect.

4. Discussion on Students' Response

We have been teaching courses that partly cover image processing for longer than a decade and a half. Image processing was taught as a part of courses such are Algorithm Complexity, Multimedia Systems, Secure Software Design, Medical Imaging, and Medical Informatics. These courses were conducted on diverse levels of studies with different complexity and structure of the student projects.

Depending on the subject, we used to point out different elements of the image processing. Somewhere the focus is on the algorithm design, somewhere on the

integration in large-scale systems, somewhere on a minimal alteration of the existing algorithms with a focus on the execution performance. For the evaluation of different programming approaches, we used the subject of Multimedia Software Systems – an electoral course in the fourth year of bachelor study. In the evaluated period, the number of students was in the range from 32 to 70 (2018 57, 2019 35, 2020 44, 2021 47, 2022 50, 2023 49, 2024 70).

The image processing software project is one of the assignments in the course, and the requirement is to implement several image processing algorithms of different complexity with minimal use of processing power and memory. For the test, the students must create an application that is comparable to Windows Forms in a .NET environment which can immediately display the result after processing is finished.

Table 3
Number of students who finished complete project.

Year	Number of enrolled students	Students who finished complete project
2018	57	45 (79%)
2019	35	22 (63%)
2020	44	29 (65%)
2021	47	39 (83%)
2022	50	41 (82%)
2023	64	58 (90%)
2024	70	61 (87%)

The only limitation that students have is related to the user interface which has to be fast and responsive and allow the display of the processed image. There is no specific request for a certain technology, besides, we advocate Microsoft .NET. On the other hand, there are no limitations to the implementation of image processing algorithms themselves.

Table 3 shows the number of enrolled students per year together with the number of students who successfully finished the entire project. Excluding the years 2019 and 2020, where the lectures were conducted in online mode, the percentage of the students who finished the projects was above 80%. During the period of online classes, this percentage dropped to less than two-thirds.

In the previous year (2023) we made a slight change in course organization by moving the image processing project to be the last in the row, as we considered it more complex. This gave the students the possibility to work on it in a period where they had fewer overall tasks during the school year. This resulted in the highest percentage of successfully finished projects at 90%. This year, we will follow the same approach, and, at the beginning of June, we will have complete data for 2024.

Data in Table 2 and Table 3 show the project distribution per technology. The projects, by the implementation technology of image processing algorithms, could be categorized into four major categories – managed, native, unsafe, and other. Projects developed in .NET and Java are considered managed code-based projects.

The projects marked as "native" are those whose algorithms are developed in various C++ environments, regardless of the technology used for the front end. In the category unsafe are these that follow the suggestion to include unsafe code blocks in managed projects. Category other is for the projects implemented in various Web technologies with different approaches considering implementations of algorithms both in the front and back end, using technologies such are various JavaScript-based frameworks.

After the students uploaded their projects in the collaborative learning platform, initially it was Moodle and later switched to Microsoft Teams, the projects were checked for performance in the demo machine to verify against the same conditions. The demo machine is an Intel-based i7-8550U running at 1.8 GHz with 4 cores and 8 logical processors, supported by 16GB of RAM. It is important to point out that all projects from the year 2018 until now are run and evaluated under the same conditions.

The Intel processors which mark ends by the sign U are not designed primarily for speed, but rather for energy efficiency. In that sense, such a computer is the perfect environment for the execution demo since the differences are better displayed.

The obvious benefit of using native and unsafe approaches can be seen in Table 6. Comparing execution time between native and unsafe approaches shows that native code runs 10 to 20 percent faster. The exact time varies depending on which moment of implementation students brought unsafe mechanisms to the project. For those that start with unsafe functions during data loading, the results are better than those that use unsafe mechanisms only for the algorithm execution.

The significant difference is between unsafe and managed code. The difference is in dozens of multiplications. **Figure 12** shows that the difference is such, that the logarithmic scale could be easily employed to display the difference. Other approaches, based on different Web technologies demonstrate the worst results in the sense of the execution time. Based on Web technology, the disadvantage is that considerable time is required to upload the source picture and then to download the results, which, makes the situation worse.

What could be seen, during the years, is that students accept the resource awareness narrative at a high percent. Besides the higher popularity of Web applications and JavaScript-based frameworks, this

approach was not the dominant choice to manage image processing problems.

Table 4
Distribution of technology used in successful student projects.

Year	Managed	Native	Unsafe	Other
2018	21	8	12	4
2019	14	2	6	1
2020	17	4	6	2
2021	10	11	13	5
2022	6	12	17	6
2023	8	17	27	6
2024	11	12	31	7

Table 5
Distribution of technology used in unfinished student projects.

Year	Managed	Native	Unsafe	Other
2018	1	5	2	4
2019	3	3	4	3
2020	7	5	3	1
2021	0	4	1	3
2022	1	6	0	2
2023	1	2	1	2
2024	4	1	2	2

Projects predicated on managed code predominated in 2019 and 2020, coinciding with the period when lectures were delivered online. With the resumption of conventional in-person lectures and the enhancement of interactive, hands-on laboratory demonstrations, there was a discernible shift in preference towards approaches that are more efficient in terms of performance.

Table 6
Average execution time by technology in benchmark machine for default image of 2000x2000 resolution in milliseconds

Approach	Simple	Convolution	Displacement
Managed C#	2054.77	6810.82	15490.77
Managed Java	2955.41	8191.38	18927.35
Native	35.41	124.77	450.20
Unsafe	39.61	146.76	559.44
Other	4850.66	11843.17	41817.93

Table 7
The average number of working days that student needed for implementation.

Year	Managed	Native	Unsafe	Other
2018	7	12	8	9
2019	11	10	10	15
2020	10	11	12	15
2021	7	13	10	11
2022	6	14	9	8
2023	8	12	9	8
2024	7	13	8	9

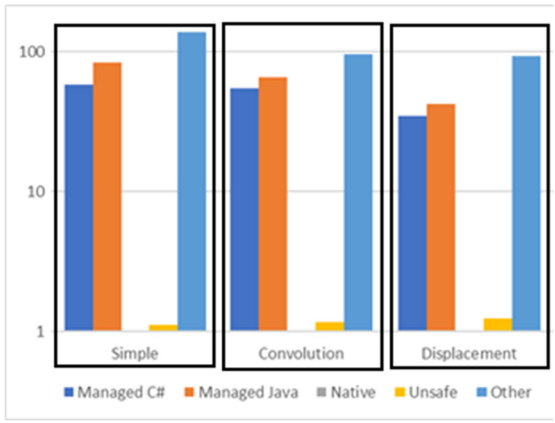

Figure 12: Comparison of relative execution times for simple, convolution, and displacement filters.

Student projects are set up for three weeks, which makes 15 days the maximal amount of time needed for the execution. During the project, students must implement several filters that belong to various categories. The filters that should be implemented are changed each year. Besides that, the overall project complexity tends to be kept, in the teachers' opinion, on the same level.

Table 7 and Figure 13 show an insight into how many days students were active on the project. These numbers represent the difference between the dates when students downloaded the assignment and the dates when the solutions were submitted. This is not the best conceivable way, since there is no exact way to prove the correctness of this approach, but, on the other hand, there is no morally acceptable way to measure how much time students spend working on their projects. Optionally the survey could be created, but the answers could be disputable either.

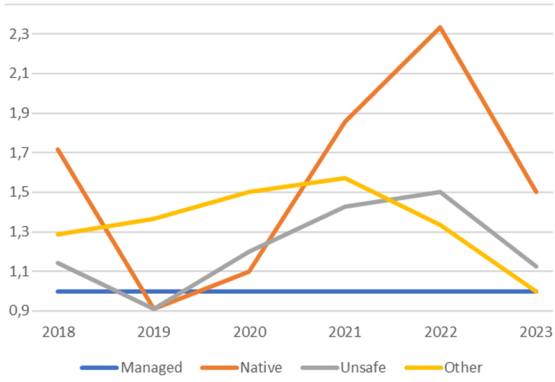

Figure 13: Comparison of relative programming effort between different programming approaches through observed years.

Nevertheless, several observations can be discerned. Primarily, students who commence their assignments late tend to opt for a managed solution or a JavaScript-based framework, as these approaches necessitate less time investment. This trend was particularly pronounced during the years 2019 and 2020 when instruction was conducted in an online format.

Most unsuccessful solutions were developed using the native code, which is logical given its demand for more time and advanced programming expertise. Additionally, it is noteworthy that the greatest failure rate was observed in the "other" category. This may be attributed to students' lack of sufficient familiarity with technologies that appeared promising, yet they lacked adequate experience to utilize them effectively.

An unsafe approach seemed like a good balance. It offers the possibility to start with the managed approach, reach some point in implementation, make the proof concept of all the algorithms, and then convert only parts of the project to the native code. Besides the approach being heavily technology-dependent and even not easily portable, it is a good example of the hybrid approach in development and how this kind of approach could bring overall benefits.

For an average of 15% more time than needed than for the projects based on managed code, the output runs only 10% slower compared to the projects in which image processing algorithms are entirely built in the native code. Ignoring years 2019 and 2020, the average time needed for the native approach is around 50% more than with the managed code.

5. Conclusion

Bringing the concepts of general resource awareness in programming is yet again important. Besides the rising popularity of fast-to-build and nice-to-look frameworks, followed by the constant increase of processing power and memory volume of all computational devices, one cannot entirely rely on the easiest solutions.

As could be seen with a managed approach in the languages based on the execution virtual machine, the two-line implementation could save time while programming, but it will result in a fifty times slower overall execution.

The use of the native environment will, of course, bring the best possible results, but the price will be significantly longer development phase. In that sense, the technology-specific solutions, like unsafe, could be an extremely good compromise between approaches. Getting 10% worse results, compared to native code, with spending around 15% more time compared to a completely managed approach could be in most cases optimal way.

However, each of the presented approaches has its pros and contras, and depending on the type of the projects, domain of usage, and the expectations regarding performance and memory usage, could be voted as optimal. In any case, future programmers must have a good overview of all the options and be aware of the appropriate use for the most valuable resource they choose – either development time or the use of the execution resources.

In this context, we posit that the incorporation of resource-awareness principles into the educational curriculum is crucial, particularly in the concluding year of study. Our research advocates for an instructional approach in image-processing education that utilizes managed code examples instead of pseudocode for algorithmic elucidation during lectures. Concurrently, laboratory exercises should employ unsafe and unmanaged code to elucidate the disparities in execution velocity, which can be significant in certain instances.

Through this paper, our objective is to convey that while numerous contemporary technologies offer considerable programming convenience and efficient development at the business layer, they should not be indiscriminately adopted as a panacea for all programming challenges. Instead, prospective developers must meticulously analyze the specific problem domain and judiciously select appropriate technologies for each component therein.

Acknowledgments

The Ministry of Science, Technological Development and Innovation of the Republic of Serbia supported this work [grant number 451-03-65/2024-03/200102].
This work is partially supported by the cost action CA 19135 CERCIRAS (Connecting Education and Research Communities for an Innovative Resource Aware Society).

References

[1] M. J. Burge, W. Burger, Digital Image Processing: An Algorithmic Introduction, Springer International Publishing AG, 2022

[2] J. R. Jensen, Introductory Digital Image Processing, Third Edition (Prentice Hall Series in Geographic Information Science), 3rd. ed., Prentice Hall, 2004.

[3] B. Jähne, Digital Image Processing: Concepts, Algorithms, and Scientific Applications, Springer London, Limited, 2013.

[4] D. Sage and M. Unser, "Teaching image-processing programming in Java," in IEEE Signal Processing

Magazine, vol. 20, no. 6, pp. 43-52, Nov. 2003, doi: 10.1109/MSP.2003.1253553.

[5] L. de O. Alves, L. F. Cruz, P. T. M. Saito, and P. H. Bugatti, "Towards Practical Computer Vision in Teaching and Learning of Image Processing Theories," 2019 IEEE Frontiers in Education Conference (FIE), Covington, KY, USA, 2019, pp. 1-7, doi: 10.1109/FIE43999.2019.9028645.

[6] Gil, P., García, G. J., Puente, S. T., Mateo, C. M., Alacid, B., & Mira, D. (2016). Teaching image and video processing with a practical cases-based methodology at the University of Alicante. In EDULEARN16 Proceedings (pp. 6067-6077). IATED.

[7] V. Monga, Y. Li, Y. C. Eldar, Algorithm Unrolling: Interpretable, Efficient Deep Learning for Signal and Image Processing, IEEE Signal Process. Mag. 38.2 (2021) 18–44. doi:10.1109/msp.2020.3016905.

[8] J. Tang, G. Liu, Q. Pan, A Review on Representative Swarm Intelligence Algorithms for Solving Optimization Problems: Applications and Trends, IEEE/CAA J. Autom. SIn. 8.10 (2021) 1627–1643. doi:10.1109/jas.2021.1004129.

[9] University of Niš, Faculty of Electronic Engineering, Course Catalog, English. URL: https://www.ni.ac.rs/en/studies-and-admission/studies/course-catalogue/courses?task=download.send&id=6594&catid=666&m=0.

[10] RGB24 pixel format for digital imaging, URL: https://www.theimagingsource.com/en-us/documentation/icimagingcontrolcpp/PixelformatRGB24.htm.

[11] Windows Forms for .NET 7 documentation. URL: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/?view=netdesktop-8.0.

[12] Managed Execution Process - .NET. URL: https://learn.microsoft.com/en-us/dotnet/standard/managed-execution-process.

[13] Bartyzel, Krzysztof. "Adaptive kuwahara filter." Signal, image and video processing 10 (2016): 663-670.

[14] Unsafe code, pointers to data, and function pointers - C#. URL: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/unsafe-code.

[15] Bitmap Image File (BMP), Version 5. URL: https://www.loc.gov/preservation/digital/formats/fdd/fdd000189.shtml.

[16] Bitmap Class (System.Drawing). URL: https://learn.microsoft.com/en-us/dotnet/api/system.drawing.bitmap?view=dotnet-plat-ext-8.0.

[17] BitmapData Class (System.Drawing.Imaging). URL: https://learn.microsoft.com/en-us/dotnet/api/system.drawing.imaging.bitmapdata?view=dotnet-plat-ext-8.0.

[18] G. Chobanyan, A Comprehensive Guide to Unsafe, Unmanaged Code and Pointers, 2023. URL: https://itnext.io/a-comprehensive-guide-to-unsafe-unmanaged-code-and-pointers-b8e143867b3e.

[19] Hiary, Hazem, et al. "Image contrast enhancement using geometric mean filter." Signal, Image and Video Processing 11 (2017): 833-840.

[20] Ibraheem, N. A., Hasan, M. M., Khan, R. Z., & Mishra, P. K. (2012). Understanding color models: a review. ARPN Journal of science and technology, 2(3), 265-275.