The final report.

Szymon Kietliński, Aleksander Karpiuk, Adam Abramczyk

1. **Introduction**

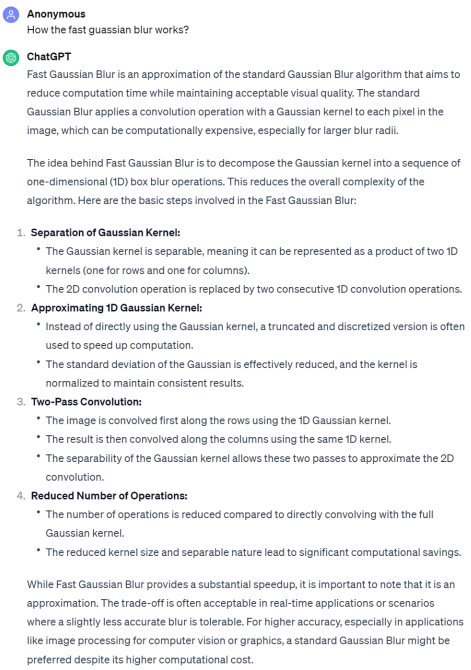
   Image processing
   GitHub repository: https://github.com/olek2303/image-processing
   Division of the responsibilities is:
   - Szymon Kietliński - Reporting, Blurring part
   - Aleksander Karpiuk - GitHub maintenance, Merging part
   - Adam Abramczyk - Work organization, Deleting part

   Our task was to find and implement, in three different programming languages, algorithms for image blurring, merging two images and deleting parts of the image. There were three main ideas: simplicity, velocity, space efficiency.
   The chosen languages:
   - C++
   - Java
   - Python.
   We decided not to create GUI for the apps and we minimize the usage of the external libraries.

2. **Algorithms**

- **Blurring**

The first idea was the obvious one - the Gaussian Blur. Then I asked chatGPT, the answer is in the screenshot below. Then I did some research on Google and found the [ref1] article and decided to implement **the fastest gaussian blur** algorithm. With the article and explanation from chat I was ready to implement this algorithm. The complexity of the algorithm is **O(n)** - this means linear. Where n is the number of pixels of the image and we assume the const strength (sigma/radius) of the blur.
For me at this part of work AI tools weren't necessary. I believe that it is better to do research, to find the algorithms, by myself and see examples and tests of it and leave AI for the implementation part.



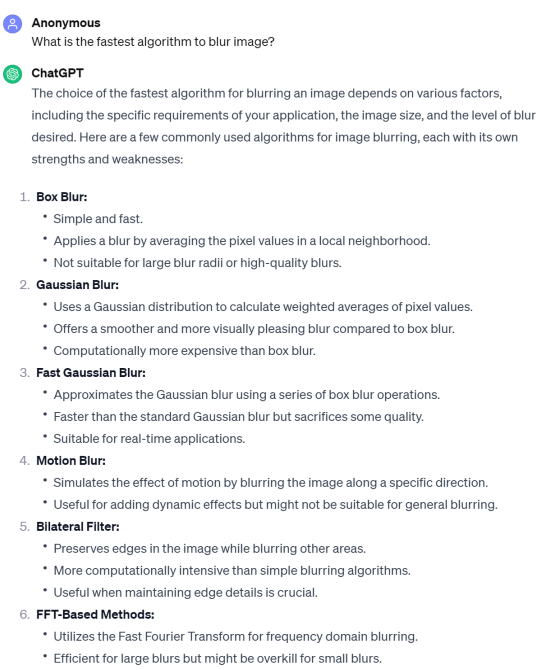
Anonymous
How the fast guassian blur works?

ChatGPT
Fast Gaussian Blur is an approximation of the standard Gaussian Blur algorithm that aims to reduce computation time while maintaining acceptable visual quality. The standard Gaussian Blur applies a convolution operation with a Gaussian kernel to each pixel in the image, which can be computationally expensive, especially for larger blur radii.

The idea behind Fast Gaussian Blur is to decompose the Gaussian kernel into a sequence of one-dimensional (1D) box blur operations. This reduces the overall complexity of the algorithm. Here are the basic steps involved in the Fast Gaussian Blur:

1. **Separation of Gaussian Kernel:**
   - The Gaussian kernel is separable, meaning it can be represented as a product of two 1D kernels (one for rows and one for columns).
   - The 2D convolution operation is replaced by two consecutive 1D convolution operations.
2. **Approximating 1D Gaussian Kernel:**
   - Instead of directly using the Gaussian kernel, a truncated and discretized version is often used to speed up computation.
   - The standard deviation of the Gaussian is effectively reduced, and the kernel is normalized to maintain consistent results.
3. **Two-Pass Convolution:**
   - The image is convolved first along the rows using the 1D Gaussian kernel.
   - The result is then convolved along the columns using the same 1D kernel.
   - The separability of the Gaussian kernel allows these two passes to approximate the 2D convolution.
4. **Reduced Number of Operations:**
   - The number of operations is reduced compared to directly convolving with the full Gaussian kernel.
   - The reduced kernel size and separable nature lead to significant computational savings.

While Fast Gaussian Blur provides a substantial speedup, it is important to note that it is an approximation. The trade-off is often acceptable in real-time applications or scenarios where a slightly less accurate blur is tolerable. For higher accuracy, especially in applications like image processing for computer vision or graphics, a standard Gaussian Blur might be preferred despite its higher computational cost.



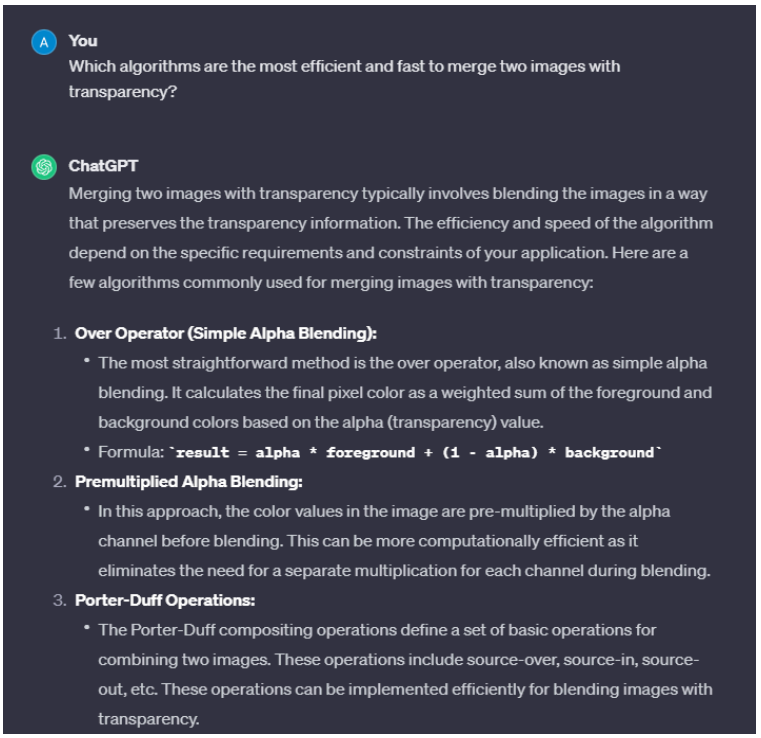
Anonymous
What is the fastest algorithm to blur image?

ChatGPT
The choice of the fastest algorithm for blurring an image depends on various factors, including the specific requirements of your application, the image size, and the level of blur desired. Here are a few commonly used algorithms for image blurring, each with its own strengths and weaknesses:

1. **Box Blur:**
   - Simple and fast.
   - Applies a blur by averaging the pixel values in a local neighborhood.
   - Not suitable for large blur radii or high-quality blurs.
2. **Gaussian Blur:**
   - Uses a Gaussian distribution to calculate weighted averages of pixel values.
   - Offers a smoother and more visually pleasing blur compared to box blur.
   - Computationally more expensive than box blur.
3. **Fast Gaussian Blur:**
   - Approximates the Gaussian blur using a series of box blur operations.
   - Faster than the standard Gaussian blur but sacrifices some quality.
   - Suitable for real-time applications.
4. **Motion Blur:**
   - Simulates the effect of motion by blurring the image along a specific direction.
   - Useful for adding dynamic effects but might not be suitable for general blurring.
5. **Bilateral Filter:**
   - Preserves edges in the image while blurring other areas.
   - More computationally intensive than simple blurring algorithms.
   - Useful when maintaining edge details is crucial.
6. **FFT-Based Methods:**
   - Utilizes the Fast Fourier Transform for frequency domain blurring.
   - Efficient for large blurs but might be overkill for small blurs.

- **Merging**

I have started my merging part of this project by asking ChatGPT, which algorithm is the best for this task.

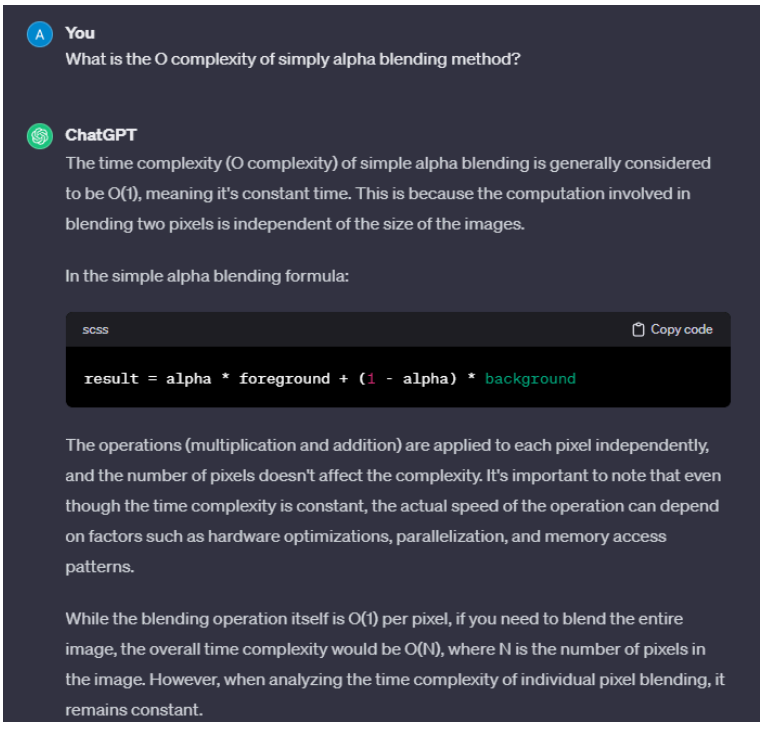


You
Which algorithms are the most efficient and fast to merge two images with transparency?

ChatGPT
Merging two images with transparency typically involves blending the images in a way that preserves the transparency information. The efficiency and speed of the algorithm depend on the specific requirements and constraints of your application. Here are a few algorithms commonly used for merging images with transparency:

1. **Over Operator (Simple Alpha Blending):**
   - The most straightforward method is the over operator, also known as simple alpha blending. It calculates the final pixel color as a weighted sum of the foreground and background colors based on the alpha (transparency) value.
   - Formula: `result = alpha * foreground + (1 - alpha) * background`
2. **Premultiplied Alpha Blending:**
   - In this approach, the color values in the image are pre-multiplied by the alpha channel before blending. This can be more computationally efficient as it eliminates the need for a separate multiplication for each channel during blending.
3. **Porter-Duff Operations:**
   - The Porter-Duff compositing operations define a set of basic operations for combining two images. These operations include source-over, source-in, source-out, etc. These operations can be implemented efficiently for blending images with transparency.

The output was very satisfying for me. I also received a formula which tells how it works. I did research about this method (Simple Alpha Blending) because it meets the project assumptions. Firstly, I read about it on Wikipedia. It helped me to understand how it works in detail. I have also asked ChatGPT about the complexity of this algorithm. I have received information that it is O(n) so I found it optimum.



You
What is the O complexity of simply alpha blending method?

ChatGPT
The time complexity (O complexity) of simple alpha blending is generally considered to be O(1), meaning it's constant time. This is because the computation involved in blending two pixels is independent of the size of the images.

In the simple alpha blending formula:

scss Copy code

```
result = alpha * foreground + (1 - alpha) * background
```

The operations (multiplication and addition) are applied to each pixel independently, and the number of pixels doesn't affect the complexity. It's important to note that even though the time complexity is constant, the actual speed of the operation can depend on factors such as hardware optimizations, parallelization, and memory access patterns.

While the blending operation itself is O(1) per pixel, if you need to blend the entire image, the overall time complexity would be O(N), where N is the number of pixels in the image. However, when analyzing the time complexity of individual pixel blending, it remains constant.

- **Deleting**

The deleting algorithm was quite simple, and big research wasn't necessary. The idea is to go via image pixels, and change them to white on the entered area.

3. Images

Blurring

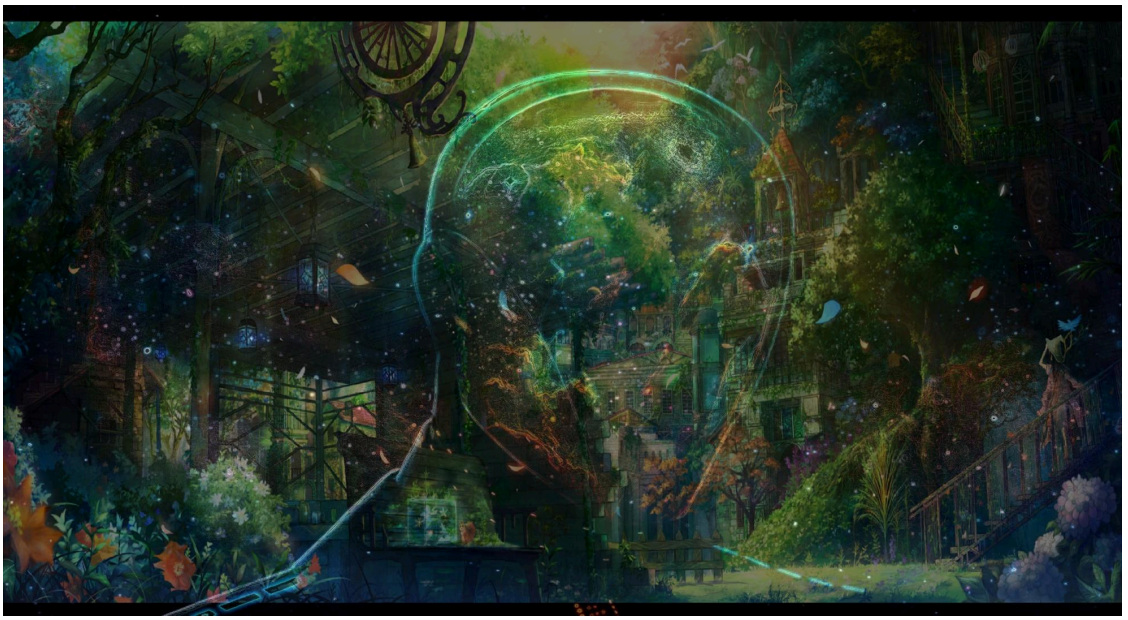
Orginal image	image info
	626 x 418 225 KB
	1980 x 1080 582 KB
	4000 x 3000 4290 KB

Blurred example (blur strength = 5)

Merging

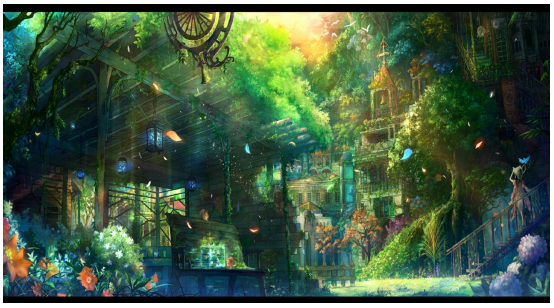
Orginal image	image info
	1980 x 1080 220 KB
	1980 x 1080 582 KB

	2434 x 1697 2670 KB
	4000 x 3000 4290 KB

Merged example (alpha = 0.5)


Deleting

Orginal image	**image info**

	626 x 418 225 KB
	1980 x 1080 582 KB
	4000 x 3000 4290 KB

Deleted example

4. Implementation of the algorithms

- Blurring

Since I found ready code [ref2] for C++ and I know this language I didn't use AI tools. I redid the existing code by myself.
Java part was completely different, especially because I have never used it. Firstly I asked chatGPT if he can rewrite the C++ code to Java but it was a complete disaster. So I started rewriting it manually, asking for the parts of codes/constructions.
The good thing was that ChatGPT was able to implement my Java code in Python and it was working. (the link for this: https://chat.openai.com/share/072c8fad-452e-48a0-8191-80c5e66ae97a)

You
w c++ mam wyrażenie int boxes[3] a w javie nie działa

ChatGPT
W języku Java deklaracja tablicy jest nieco inna niż w C++. W Javie należy użyć słowa kluczowego `int[]` w celu zadeklarowania tablicy. Przykładowo:

```java
int[] boxes = new int[3];
```

Powyższy kod utworzy tablicę o nazwie `boxes` zawierającą trzy elementy typu `int`. Pamiętaj, że w Javie indeksy tablic zaczynają się od 0, więc indeksy dostępne dla tej tablicy to 0, 1 i 2.

Możesz również zainicjować tablicę od razu z wartościami, na przykład:

```java
int[] boxes = {1, 2, 3};
```

To utworzy tablicę `boxes` z trzema elementami o wartościach 1, 2 i 3.

Ogólnie zasady korzystania z tablic w Javie są podobne do C++, ale składnia jest nieco inna.

You
okej a czy w javie muszę niszczyc takie obiekty jakimś delete?

ChatGPT
W Javie nie musisz ręcznie usuwać obiektów, tak jak to się robi w C++ za pomocą operatora `delete`. W Javie zarządzanie pamięcią jest zautomatyzowane i odbywa się za pomocą mechanizmu zwalniania pamięci zwanej Garbage Collectorem (GC).

W skrócie, kiedy obiekt w Javie przestaje być dostępny, ponieważ nie ma do niego referencji, Garbage Collector automatycznie usuwa ten obiekt i zwalnia zasoby pamięci, które zajmował. Programista nie musi martwić się o ręczne zwalnianie pamięci.

Ważne jest, aby pamiętać o kilku rzeczach:

1. **Garbage Collector działa automatycznie:** Nie musisz wywoływać żadnej metody zwalniającej pamięć, to zadanie zostanie wykonane przez Garbage Collector.
2. **Zwalnianie zasobów związanych z obiektami:** Chociaż pamięć jest zwalniana automatycznie, inne zasoby (np. otwarte pliki, połączenia z bazą danych) mogą wymagać ręcznego zwolnienia, używając odpowiednich metod (np. `close()` dla plików czy połączeń z bazą danych).
3. **Nullowanie referencji:** Aby pomóc Garbage Collectorowi, możesz nullować referencje do obiektów, gdy już ich nie potrzebujesz. To może przyspieszyć proces zwalniania pamięci, zwłaszcza w przypadku długotrwałych programów.

Przykład:

```java
SomeObject obj = new SomeObject();
// ...
obj = null; // Nullowanie referencji, aby pomóc Garbage Collectorowi
```

W skrócie, w Javie pamięć zarządzana jest automatycznie, co eliminuje potrzebę manualnego zwalniania pamięci, jak to ma miejsce w C++.

You
inaczej pokaże ci jak ja to mam:

```
 Raster raster = im.getData();
    int[] intPixels = ((DataBufferInt) raster.getDataBuffer()).getData();

    byte[] pixels = new byte[intPixels.length];
    for (int i = 0; i < intPixels.length; i++) {
      pixels[i] = (byte) intPixels[i];
    }
//modyfikuje pixels i teraz chcę je zapisać
```

ChatGPT
Dzięki za dodatkowe wyjaśnienie. Jeśli masz dane pikseli w formie tablicy bajtów (`byte[]`) i chcesz je zapisać z powrotem do obiektu `BufferedImage`, ale masz problem ze zrozumieniem, jak to zrobić, oto kod, który może ci pomóc:

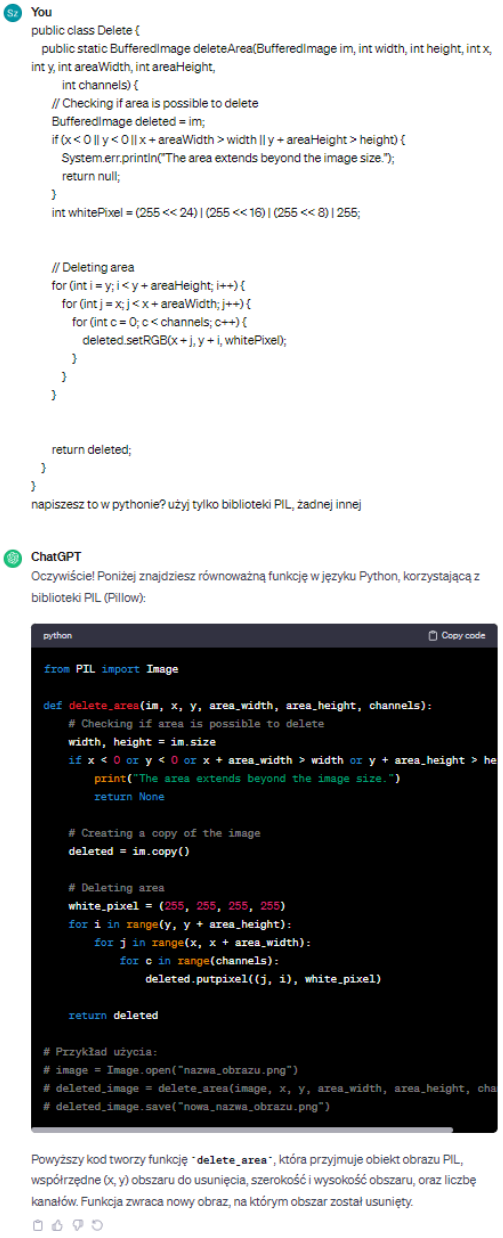
```java
import java.awt.image.BufferedImage;
import java.awt.image.DataBufferByte;

public class ImageProcessing {
    public static void main(String[] args) {
        BufferedImage image = // ... Twój obraz
        byte[] pixels = // ... Twoja tablica bajtów

        // Ustaw zmodyfikowane piksele z powrotem do obrazu
        setModifiedPixels(image, pixels);
    }

    // Metoda ustawiająca zmodyfikowane piksele z powrotem do obrazu
    private static void setModifiedPixels(BufferedImage image, byte[] m
        // Pobierz dane obrazu
        DataBufferByte dataBuffer = (DataBufferByte) image.getRaster().
        byte[] imagePixels = dataBuffer.getData();

        // Skopiuj zmodyfikowane piksele do danych obrazu
        System.arraycopy(modifiedPixels, 0, imagePixels, 0, modifiedPix

        // Przerysuj obraz (jeśli potrzebne)
        image.setData(image.getRaster());
    }
}
```

- **Merging**

When it came to implementing this algorithm to the project I saw an article which showed me that it is very important to work on channels. I have previously had a formula of this algorithm so I could start with the implementation. After many tries with half-working code I decided to ask ChatGPT about the full code of this algorithm in C++. I received a working version for images with different resolutions after many tries with Chat (https://chat.openai.com/share/18f27238-efe1-46bf-904a-4331f17ab703). When it came to implement this algorithm in Java I have started on my own. I figured out that I do not need any library for the image processing. I started searching for information about getting channels in Java. Later I asked ChatGPT to convert the code from C++ to Java and it worked (https://chat.openai.com/share/3c49be6e-9ad5-46d4-98ab-45ceb288b915). With Python the situation was the same, but I had to ask ChatGPT about the optimization of the code because it took so much time for Python to process this algorithm.

- **Deleting**

I used chatGPT to generate the Python code from Java. It works perfectly, no changes were needed.



You

```
public class Delete {
  public static BufferedImage deleteArea(BufferedImage im, int width, int height, int x,
int y, int areaWidth, int areaHeight,
      int channels) {
    // Checking if area is possible to delete
    BufferedImage deleted = im;
    if (x < 0 || y < 0 || x + areaWidth > width || y + areaHeight > height) {
      System.err.println("The area extends beyond the image size.");
      return null;
    }
    int whitePixel = (255 << 24) | (255 << 16) | (255 << 8) | 255;


    // Deleting area
    for (int i = y; i < y + areaHeight; i++) {
      for (int j = x; j < x + areaWidth; j++) {
        for (int c = 0; c < channels; c++) {
          deleted.setRGB(x + j, y + i, whitePixel);
        }
      }
    }


    return deleted;
  }
}
```

napiszesz to w pythonie? użyj tylko biblioteki PIL, żadnej innej

ChatGPT

Oczywiście! Poniżej znajdziesz równoważną funkcję w języku Python, korzystającą z biblioteki PIL (Pillow):

python — Copy code

```python
from PIL import Image

def delete_area(im, x, y, area_width, area_height, channels):
    # Checking if area is possible to delete
    width, height = im.size
    if x < 0 or y < 0 or x + area_width > width or y + area_height > he
        print("The area extends beyond the image size.")
        return None

    # Creating a copy of the image
    deleted = im.copy()

    # Deleting area
    white_pixel = (255, 255, 255, 255)
    for i in range(y, y + area_height):
        for j in range(x, x + area_width):
            for c in range(channels):
                deleted.putpixel((j, i), white_pixel)

    return deleted

# Przykład użycia:
# image = Image.open("nazwa_obrazu.png")
# deleted_image = delete_area(image, x, y, area_width, area_height, cha
# deleted_image.save("nowa_nazwa_obrazu.png")
```

Powyższy kod tworzy funkcję `delete_area`, która przyjmuje obiekt obrazu PIL, współrzędne (x, y) obszaru do usunięcia, szerokość i wysokość obszaru, oraz liczbę kanałów. Funkcja zwraca nowy obraz, na którym obszar został usunięty.

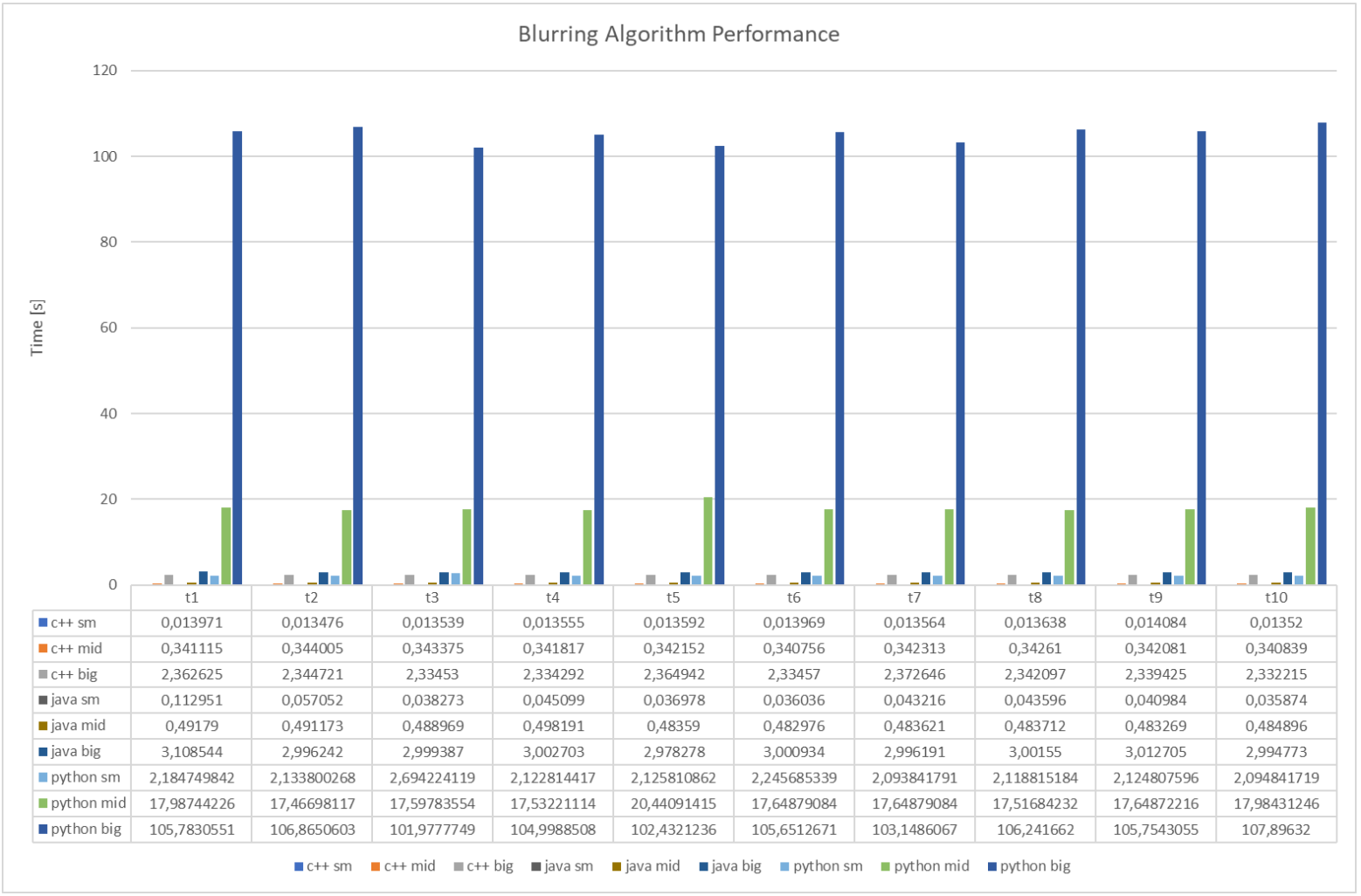
5. Analysis of the results

- Blurring

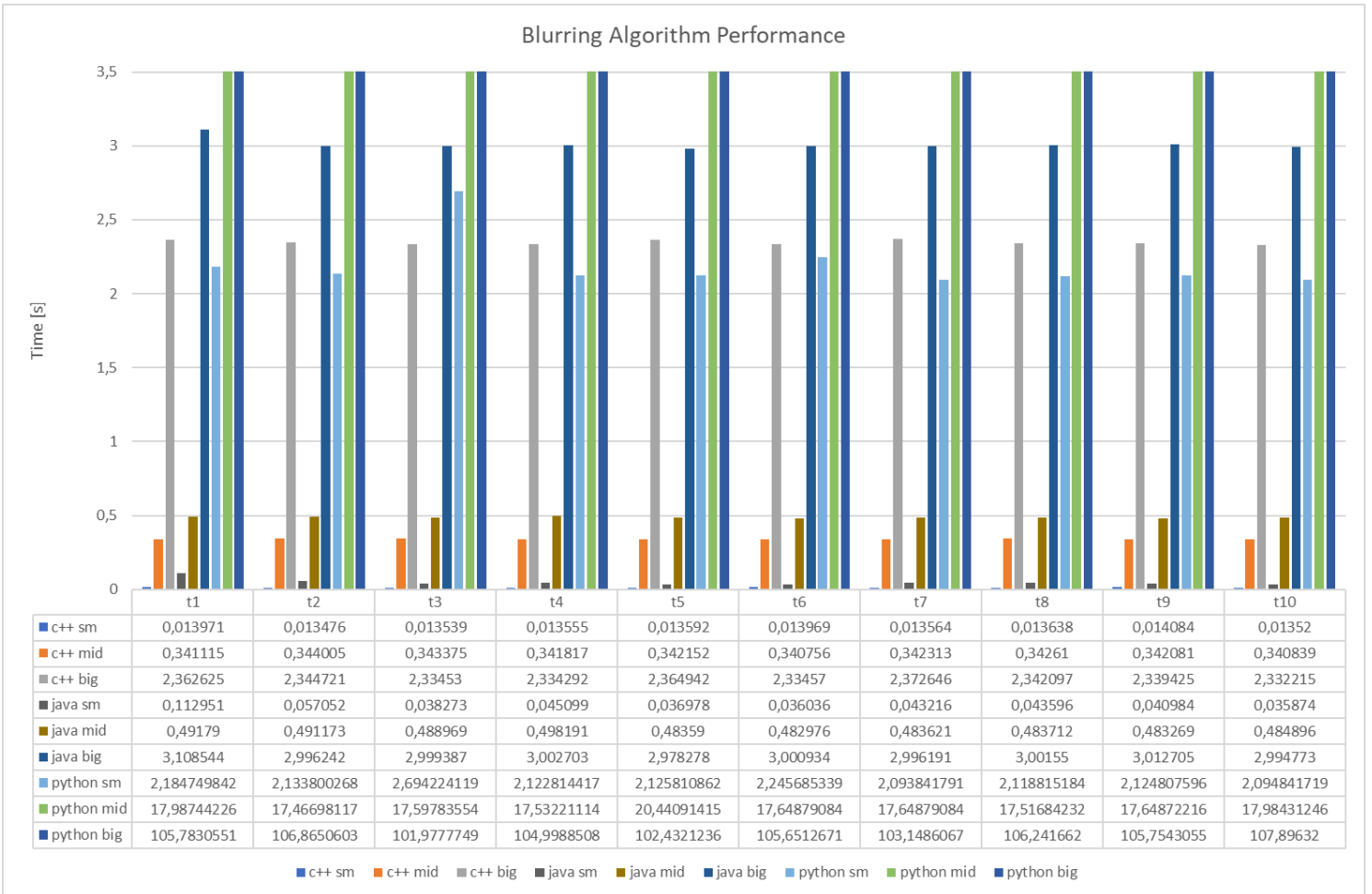
C++ is the fastest, avg times:
small: 0,0136908 s
mid: 0,3421063 s
big: 2,3462063 s

Java:
small: 0,0490059 s -> 3.575 times slower than C++
mid: 0,4872187 s -> 1.424 times slower than C++
big: 3,0091307 s -> 1.282 times slower than C++

Python:
small: 2,193939114 s -> 160.491 times slower than C++
mid: 17,94728429 s ->52.430 times slower than C++
big: 104,998850 s -> 44.733 times slower than C++



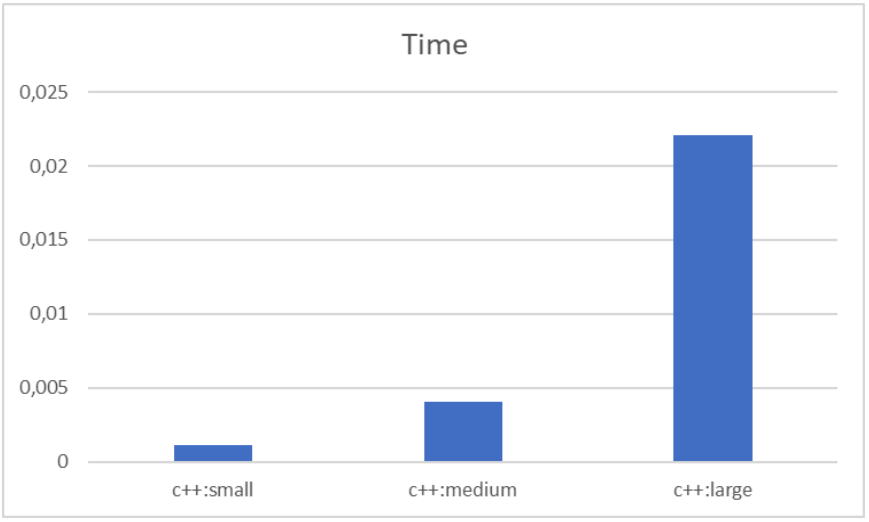
	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10
c++ sm	0,013971	0,013476	0,013539	0,013555	0,013592	0,013969	0,013564	0,013638	0,014084	0,01352
c++ mid	0,341115	0,344005	0,343375	0,341817	0,342152	0,340756	0,342313	0,34261	0,342081	0,340839
c++ big	2,362625	2,344721	2,33453	2,334292	2,364942	2,33457	2,372646	2,342097	2,339425	2,332215
java sm	0,112951	0,057052	0,038273	0,045099	0,036978	0,036036	0,043216	0,043596	0,040984	0,035874
java mid	0,49179	0,491173	0,488969	0,498191	0,48359	0,482976	0,483621	0,483712	0,483269	0,484896
java big	3,108544	2,996242	2,999387	3,002703	2,978278	3,000934	2,996191	3,00155	3,012705	2,994773
python sm	2,184749842	2,133800268	2,694224119	2,122814417	2,125810862	2,245685339	2,093841791	2,118815184	2,124807596	2,094841719
python mid	17,98744226	17,46698117	17,59783554	17,53221114	20,44091415	17,64879084	17,64879084	17,51684232	17,64872216	17,98431246
python big	105,7830551	106,8650603	101,9777749	104,9988508	102,4321236	105,6512671	103,1486067	106,241662	105,7543055	107,89632

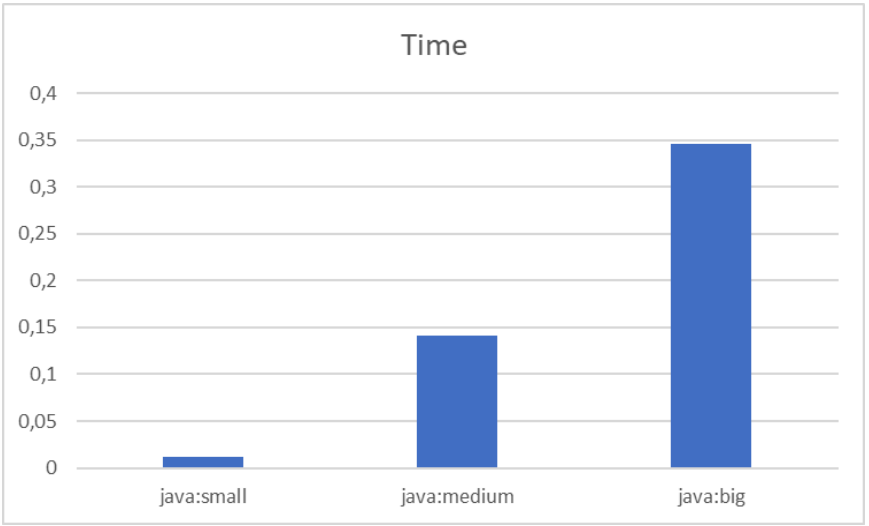


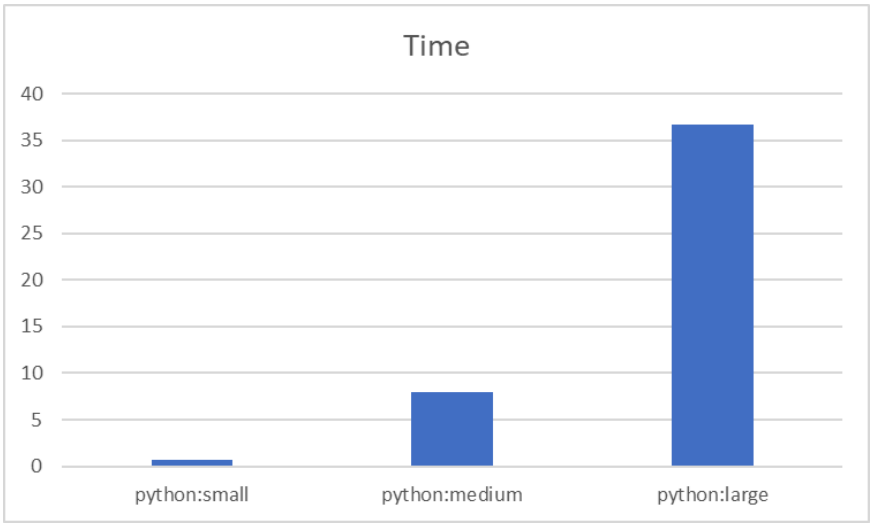
	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10
c++ sm	0,013971	0,013476	0,013539	0,013555	0,013592	0,013969	0,013564	0,013638	0,014084	0,01352
c++ mid	0,341115	0,344005	0,343375	0,341817	0,342152	0,340756	0,342313	0,34261	0,342081	0,340839
c++ big	2,362625	2,344721	2,33453	2,334292	2,364942	2,33457	2,372646	2,342097	2,339425	2,332215
java sm	0,112951	0,057052	0,038273	0,045099	0,036978	0,036036	0,043216	0,043596	0,040984	0,035874
java mid	0,49179	0,491173	0,488969	0,498191	0,48359	0,482976	0,483621	0,483712	0,483269	0,484896
java big	3,108544	2,996242	2,999387	3,002703	2,978278	3,000934	2,996191	3,00155	3,012705	2,994773
python sm	2,184749842	2,133800268	2,694224119	2,122814417	2,125810862	2,245685339	2,093841791	2,118815184	2,124807596	2,094841719
python mid	17,98744226	17,46698117	17,59783554	17,53221114	20,44091415	17,64879084	17,64879084	17,51684232	17,64872216	17,98431246
python big	105,7830551	106,8650603	101,9777749	104,9988508	102,4321236	105,6512671	103,1486067	106,241662	105,7543055	107,89632

- **Merging**

c++ with small image	0,00112	java with small image	0,011398	python with small image	0,719767
c++ with medium image	0,004053	java with medium image	0,141498	python with medium image	7,92367
c++ with large image	0,02209	java with big image	0,345667	python with large image	36,71813

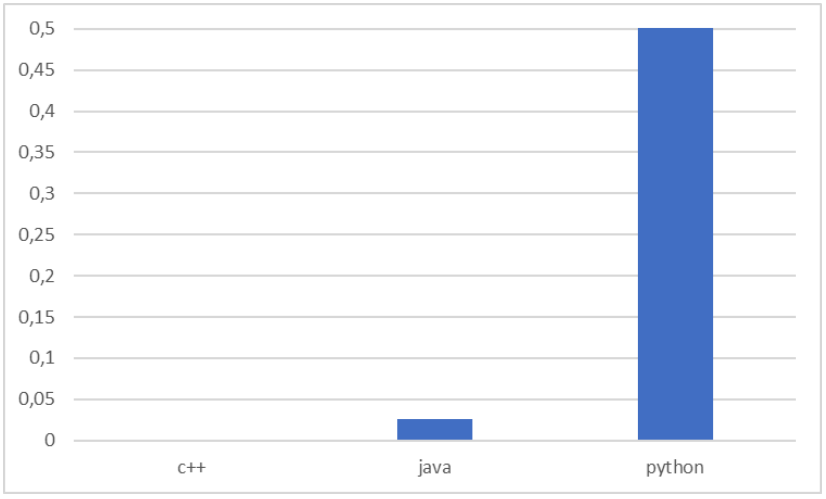






- **Deleting**

Starting coordinates were always the same (10,10) and deleting area was always 300 x 400.
C++ is the fastest, avg times:
small: 0,000394 s
mid: 0,000396 s
big: 0,000395 s
Java:
small: 0,025389 s
mid: 0,026412 s
big: 0,026687 s
Python:
small: 0,5014829635620117 s
mid: 0,404585599899292 s
big: 0,4205665588378906 s
As we can see, the picture size does not affect the execution time. The execution time depends on the size of the deleting area.



- **Summary**

The performance results generally meet expectations, C++ is the fastest because of the fact it is compiled. Java is also fast and compiled but it's required to run on the Java virtual machine. Meanwhile Python is the fastest to write, the most idiotproof, the simplest, doesn't require specifying types of variables but it's also really slow, especially in terms of the loops.

6. Conclusion

In conclusion AI tools may be really helpful with implementing parts of codes in the languages that we don't know really well. But we need to remember that usually the generated code is not the most effective, prettiest and the shortest possible. It also usually has bugs.
AI is also helpful with understanding the differences between specific constructions in programming languages.
It is important to always specify which external library we want to use/don't use etc. when asking a query to AI.

About this specific project, it is possible to write a small and quite effective app without using external libraries but the production process is slower. The pros is that the programmers better understand the whole code because they had to write it.

7. Bibliography

[Ref 1] - The fastest Guassian Blur - https://blog.ivank.net/fastest-gaussian-blur.html

[Ref 2] - The fastest Guassian Blur in C++ - https://gist.github.com/bfraboni/946d9456b15cac3170514307cf032a27

https://en.wikipedia.org/wiki/Alpha_compositing

https://4programmers.net/C/Artykuły/Alpha_Blending_-_Czyli_półprzezroczysty_obrazek

https://reintech.io/blog/java-image-processing-manipulating-analyzing-images