



**TRIBHUVAN UNIVERSITY  
INSTITUTE OF ENGINEERING  
THAPATHALI CAMPUS**

**A Project Report  
On  
Image to ASCII art**

**Submitted By:**  
Kushal Joshi (THA081BCT015)  
Diwen Baniya (THA081BCT010)  
Parth KC (THA081BCT019)

**Submitted To:**  
Department of Electronics and Computer Engineering  
Thapathali Campus  
Kathmandu, Nepal

In partial fulfillment for the award of the Bachelor's Degree in Electronics and  
Communication Engineering

**Under the Supervision of**  
Prajwal Pakka

March, 2025

## DECLARATION

We hereby declare that the report of the project entitled “**Image to ascii art**” which is being submitted to the **Department of Electronics and Computer Engineering, IOE, Thapathali Campus**, in the partial fulfillment of the requirements for the award of the Degree of Bachelor of Engineering in **Electronics and Communication Engineering**, is a report of the work carried out by us. The materials contained in this report have not been submitted to any University or Institution for the award of any degree and we are the only author of this complete work and no sources other than the listed here have been used in this work.

Kushal Joshi (THA081BCT015)

---

Diwen Baniya (THA081BCT010)

---

Parth KC (THA081BCT019)

---

**Date:** March, 2025

## **CERTIFICATE OF APPROVAL**

The undersigned certify that they have read and recommended to the **Department of Electronics and Computer Engineering, IOE, Thapathali Campus**, a minor project work entitled “**Image to ASCII art**” submitted by **Kushal Joshi, Diwen Baniya, Parth KC** in partial fulfillment for the award of Bachelor’s Degree in Electronics and Communication Engineering. The Project was carried out under special supervision and within the time frame prescribed by the syllabus.

We found the students to be hardworking, skilled and ready to undertake any related work to their field of study and hence we recommend the award of partial fulfillment of Bachelor’s degree of Electronics and Communication Engineering.

---

Project Supervisor

Prajwal Pakka

Department of Electronics and Computer Engineering, Thapathali Campus

March, 2025

## **COPYRIGHT**

The author has agreed that the library, Department of Electronics and Computer Engineering, Thapathali Campus, may make this report freely available for inspection. Moreover, the author has agreed that the permission for extensive copying of this project work for scholarly purpose may be granted by the professor/lecturer, who supervised the project work recorded herein or, in their absence, by the head of the department. It is understood that the recognition will be given to the author of this report and to the Department of Electronics and Computer Engineering, IOE, Thapathali Campus in any use of the material of this report. Copying of publication or other use of this report for financial gain without approval of the Department of Electronics and Computer Engineering, IOE, Thapathali Campus and author's written permission is prohibited.

Requests for permission to copy or to make any use of the material in this project in whole or part should be addressed to department of Electronics and Computer Engineering, IOE, Thapathali Campus.

## **ACKNOWLEDGEMENT**

We would like to express our sincere gratitude to everyone who contributed to the successful completion of our C programming project, "Image to ASCII art". This project would not have been possible without the guidance, support, and encouragement of many individuals.

First and foremost, we extend our heartfelt thanks to our C Programming instructor, Prajwal Pakka, for providing us with the opportunity to work on this project and for their invaluable guidance throughout the process. Their expertise and constructive feedback helped us overcome challenges and improve our understanding of programming concepts. We are also thankful to our colleagues and peers for their constant support, motivation, and collaborative spirit. Their suggestions and insights played a significant role in refining our project.

Lastly, we would like to acknowledge the online resources, textbooks, and tutorials that served as references during the development of this project. These resources were instrumental in enhancing our knowledge and skills in C programming. This project has been a great learning experience, and we are grateful to everyone who contributed directly or indirectly to its completion.

Kushal Joshi (THA081BCT015)

Diwen Baniya (THA081BCT010)

Parth KC (THA081BCT019)

## ABSTRACT

ASCII Art is a technique for creating artwork using text characters. This project, developed in C, converts digital images into ASCII representations. Its purpose is to explore the artistic potential of programming while gaining experience in image manipulation with C. By transforming photos into unique text-based visuals, the project enhances understanding of image processing in C. It offers a creative and engaging way to generate text-based images. Utilizing the `stb_image` library, the program loads an image from disk, processes its pixel data, and maps it to ASCII characters to maintain its visual essence. The final output is saved as a `.txt` file.

*Keywords: C programming, image processing, pixels.*

## **Table of Contents**

<b>DECLARATION</b>	<b>i</b>
<b>CERTIFICATE OF APPROVAL</b>	<b>ii</b>
<b>COPYRIGHT</b>	<b>iii</b>
<b>ACKNOWLEDGEMENT</b>	<b>iv</b>
<b>ABSTRACT</b>	<b>v</b>
<b>Table of Contents</b>	<b>vi</b>
<b>List of Figures</b>	<b>ix</b>
<b>List of Abbreviations</b>	<b>x</b>
<b>1. INTRODUCTION</b>	<b>1</b>
1.1 Background	1
1.2 Motivation	2
1.3 Problem Definition	2
1.4 Project Objectives	2
1.5 Project Scope and Applications	3
1.6 Report Organization	3
<b>2. LITERATURE REVIEW</b>	<b>4</b>
2.1 Types of Image File Formats	4
2.2 The history and development of the c programming language	4
<b>3. REQUIREMENT ANALYSIS</b>	<b>5</b>
3.1 Hardware Requirements:	5
3.1.1 Processor	4
3.1.2 Memory (RAM)	4
3.1.3 Storage	4
3.1.4 Operating System	5

3.2	Feasibility Study	5
3.2.1	Technical Feasibility	5
3.2.2	Operational Feasibility:	5
3.2.3	Economic Feasibility:	5
3.2.4	Schedule Feasibility:	5
3.2.5	Legal and Ethical Feasibility:	5
<b>4.</b>	<b>SYSTEM ARCHITECTURE AND METHODOLOGY</b>	<b>6</b>
4.1	Block Diagram/Architecture	6
		6
4.2	Methodology	6
<b>5.</b>	<b>RESULTS AND ANALYSIS</b>	<b>11</b>
<b>6.</b>	<b>FUTURE ENHANCEMENT</b>	<b>13</b>
	<b>CONCLUSION</b>	<b>15</b>
	<b>REFERENCES</b>	<b>16</b>



## **List of Abbreviations**

IOE     Institute of Engineering

JPG     Joint Photographic Experts Group

PNG     Portable Network Graphics

stb:     Sean T. Barrett

ASCII:   American Standard Code for Information Interchange

RGB     Red Green Blue

# 1. INTRODUCTION

## 1.1 Background

A programming language is a set of rules and symbols that allow humans to communicate instructions to a computer, essentially translating complex ideas into a format the machine can understand and execute, enabling the creation of software, applications, and websites through written code; different languages offer varying levels of abstraction, with "high-level" languages being more user-friendly and "low-level" languages providing greater control over hardware operations. Key points about programming languages: The C programming language is a powerful and efficient language known for its low-level system access and portability. It is widely used in system programming, embedded systems, and performance-critical applications. By developing an Image to ASCII art converter in C, we can explore core programming concepts such as input/output handling, control structures, functions, and error management.

Function: To provide a structured way for programmers to write instructions that a computer can interpret and follow.

Syntax and Semantics: Each language has its own syntax (structure of code) and semantics (meaning of the code).

Types: Programming languages can be categorized as low-level (closer to machine language) or high-level (more human-readable).

Examples: Popular programming languages include Python, Java, C++, JavaScript, C#, and PHP.

In this project we used the C programming language to create a program to convert an image to its equivalent ASCII art using the `stb_image` library.

## **Motivation**

This project aims to explore the creative intersection of art and technology by converting digital images into ASCII art. ASCII art, a form of visual representation using text characters, allows for a unique way of interpreting and displaying images, emphasizing simplicity and creativity. By developing this tool, We seek to create a program that is fun to play around with allowing user to create ASCII art easily in short amount of time.

### **1.2 Problem Definition**

The primary objective of this project is to read an image from the disk, process its pixel data, and convert it into an ASCII art representation. The resulting ASCII artwork will then be saved as a `.txt` file on the disk. To accomplish this, the project utilizes the `stb_image` library, which facilitates image loading and processing. By mapping pixel values to corresponding ASCII characters, the program aims to preserve the visual essence of the original image while presenting it in a text-based format.

### **1.3 Project Objectives**

Our goal is to develop an art creation program capable of converting existing images into ASCII art while preserving the essence of the original image. We aim to design a program that is enjoyable to use and produces visually appealing results for the user.

Through this project, we intend to achieve the following objectives:

- The program should be able to read an image from disk and process it effectively.
- It should generate ASCII art that closely resembles the original image.
- The program must allow users to save the output easily.
- The ASCII conversion should consider key properties of the original image to maintain its visual integrity.

By working on this project, we have gained fundamental knowledge of handling and manipulating various image file formats.

## 1.4 Project Scope and Applications

This project emphasizes the use of the `stb_image` library for processing and manipulating image files. Through this work, we have gained experience in handling `.jpeg` and `.png` files in various ways. The concepts explored in this project will provide a foundation for novice programmers to apply when working with images in future projects. Since image processing is a common task in programming, the knowledge acquired here will be highly beneficial for future development.

## 1.5 Report Organization

This report is structured into multiple sections to systematically present the project development process:

- Chapter 1: Introduction – Covers the background, motivation, problem definition, objectives, scope, and organization of the report.
- Chapter 2: Literature Review – Discusses the fundamentals of image file formats, such as JPEG and PNG, and the history and evolution of the C programming language.
- Chapter 3: System Architecture and Methodology – Explains the project design, including system components, block diagrams, and development tools.
- Chapter 4: Implementation Details – Provides a detailed explanation of the program structure, function implementations, and error-handling mechanisms.
- Chapter 5: Results and Analysis – Presents the output of the project, along with performance analysis and validation of the implemented system.
- Chapter 6: Future Enhancements – Suggests potential improvements and additional features that could enhance the image-to-ASCII converter's capabilities.
- Chapter 7: Conclusion – Summarizes the project, highlighting the key takeaways and its impact.
- Chapter 8: Appendices – Includes additional materials such as code snippets, sample images, and relevant data.
- Chapter 9: References – Lists the sources used in research and development

## 2. LITERATURE REVIEW

This project's literature review covers the following key topics:

- Different types of image file formats, such as JPEG and PNG.
- The history and evolution of the C programming language.

### 2.1 Types of Image File Formats

An image file format defines how digital images are stored and structured. Various formats, including JPEG, PNG, and GIF, are commonly used. Until 2022, most image formats were designed for storing 2D images rather than 3D ones. These formats may use either compressed or uncompressed data storage methods.

#### JPEG

The JPEG (Joint Photographic Experts Group) format, identified by the extensions `.jpg` and `.jpeg`, is the most widely used lossy compression format for still images. It is particularly effective for photographs, but when applied to images that require high sharpness, such as diagrams or charts, the quality may degrade. Technically, JPEG is a compression standard rather than a file format.

#### PNG

The PNG (Portable Network Graphics) format, often pronounced "ping," uses lossless compression while supporting higher color depths than GIF. It is more efficient and offers full alpha transparency, making it ideal for images requiring high quality and clear backgrounds. PNG is widely supported across all major web browsers.

### 2.2 The History and Development of the C Programming Language

The C programming language, developed by Dennis Ritchie at Bell Labs in the early 1970s, emerged from efforts to create the Unix operating system. Seeking a portable, efficient alternative to assembly language, Ritchie built upon Ken Thompson's earlier B language, introducing features like data typing and structured programming. C's simplicity, combined with low-level memory access, made it ideal for system programming.

In 1978, Ritchie and Brian Kernighan published *The C Programming Language* (K&R), standardizing the language and fueling its adoption. The American National Standards Institute (ANSI) formalized C in 1989 (C89), followed by ISO updates (C99, C11), ensuring cross-platform compatibility. C's influence is profound: it became the foundation for languages like C++, Java, and Python. Its efficiency

keeps it vital in operating systems (Linux, Windows kernels), 1978 K&R and Dennis Ritchie, often referred to as K&R C. It served as a embedded systems, and performance-critical applications. By balancing abstract with hardware control, C

bridged high-level programming and machine efficiency, shaping modern computing. Today, it remains a cornerstone of software development, education, and systems engineering, underscoring its enduring relevance

## **2. REQUIREMENT ANALYSIS**

### **2.1 Hardware Requirements:**

#### **3.1.1 Processor**

Any modern processor should be able to run this program with ease as this is a fairly simple program which does not require a lot of computational power.

#### **3.1.2 Memory (RAM)**

A minimum of 512MB is recommended to ensure smooth execution of the program.

#### **3.1.3 Storage**

Storage requirement for the program stems from the required input image to run the program and the saved output ascii art. Storage should be decided considering the input image.

#### **3.1.4 Operating System**

Most of the operation systems that support image handling should be able to run this program with ease. For example windows, Linux and MacOS all will be able to run this program as C programming language is a very versatile language.

### **2.2 Feasibility Study**

#### **3.2.1 Technical Feasibility**

The project is technically feasible as it uses standard C libraries and does not require any specialized hardware or software. The program can be developed and executed on any system with a C compiler and sufficient computational resources.

### **3.2.2 Operational Feasibility:**

The program is executed through the terminal. It requires the user to input file name or file destination in the terminal, But the process is not too complex. First time users may have trouble using this program but it's a fairly simple to use program.

### **3.2.3 Economic Feasibility:**

The project does not require any additional financial investment beyond the cost of a standard computer system and a C compiler, which are widely available and often free.

### **3.2.4 Schedule Feasibility:**

The project can be completed within a reasonable timeframe, as we use standard libraries we are able to reduce the required time by a significant amount of time. The program also is very time efficient, and it does not require much time to execute.

### **3.2.5 Legal and Ethical Feasibility:**

The project does not involve any legal or ethical issues. As it is only a tool for image processing, even if the user uses a copyrighted image which they do not have permission to use in this program. It is not the responsibility of us makers.

## **3. SYSTEM ARCHITECTURE AND METHODOLOGY**

### **4.1. SYSTEM ARCHITECTURE**

The Image-to-ASCII Art Conversion system is designed to take a digital image and convert it into a text-based representation using ASCII characters. This system is structured around several key components that work together to achieve the desired output. The architecture is modular, where each module performs a specific function, contributing to the overall process of converting an image into ASCII art. The system architecture can be broken down into the following components:

4.1 Sub-heading 1 – System Architecture1. Image Input Module:  
○ Function: The first step in the system is accepting the input image. The system uses the command line interface (CLI), where the user specifies the path to file as an argument when executing the program.

○ How It Works: The program accepts the image file path as a command-line argument. It loads the image using the stb\_image library, which can load images in various formats like PNG, JPEG, BMP, etc.

○ Input Handling: The program ensures that the image file exists, and if not, it displays an appropriate error message. Once loaded, the system extracts essential details such as the image's width, height, and number of color channels(RGB).

## 2. Image Processing Module:

○ Function: This module processes the image data by converting the RGB values to grayscale and prepares the pixel information for ASCII conversion.

○ How It Works:

■ Each pixel's RGB values are retrieved from the image.

■ The system then performs a conversion from sRGB (standard RGB) to linear RGB using a gamma correction algorithm. The formula used adjusts the pixel values to reflect a more accurate linear light intensity.

■ After linearization, the luminance of each pixel is calculated using the formula:

$$\text{Luminance} = 0.2126 \cdot R + 0.7152 \cdot G + 0.0722 \cdot B$$

$\text{BLuminance} = 0.2126 \cdot R + 0.7152 \cdot G + 0.0722 \cdot B$  This step is crucial because the luminance value represents the perceived brightness of a pixel, which is essential for the ASCII conversion.



- The luminance is then gamma-compressed back to sRGB to fit within the usual display range.

### 3. Grayscale Conversion and Mapping to ASCII:

- Function: This module maps the grayscale value of each pixel to a corresponding ASCII character, creating the text-based representation of the image.

- How It Works:

- Once the grayscale value for each pixel is computed, it is mapped to a character in the ASCII character set. A darker pixel will be mapped to a denser character (e.g., '@' or '#'), while a lighter pixel will be mapped to a lighter character (e.g., '.' or ' ').

- The character mapping process follows a predefined set of characters, arranged from darkest to lightest. The grayscale value is scaled to fit the range of characters available in the set.

- This mapping process turns the image into a grid of ASCII characters, preserving the image's structure but using characters instead of pixels.

### 4. Output Module:

- Function: The final ASCII representation of the image is output as a text file.

- How It Works:

- The ASCII characters are written to a .txt file. Each row in the text file corresponds to a row of pixels in the original image, and each pixel's grayscale value is represented by a single character.

- The file is saved with the name `ascii_art.txt` or a custom name, depending on user preferences.

### 5. Error Handling and Validation:

- Function: The system includes basic error handling to ensure smooth execution.
- How It Works:
  - The program checks if the input image file is valid and can be opened.
  - If the image cannot be loaded or the file is invalid, the system will output an error message and terminate gracefully.
  - The system also checks that memory allocations for image data and ASCII output are successful. If memory allocation fails, the program will notify the user and exit.

### 3.3 Tools and Environment

1. IDE: Visual Studio Code (VS Code): This lightweight and highly customizable code editor is used for writing and editing the C code. VS Code Provides excellent support for C/C++ with extensions like C/C++ IntelliSense, CMake Tools, and Debugger to help streamline development.
2. Compiler: MinGW GCC: This compiler is used to compile the C code into an executable. It is a suitable tool for compiling code on Windows, and it is lightweight and easy to configure.
3. Libraries:
  - stb\_image: The stb\_image library is used to load image files. It supports multiple formats, such as PNG, JPEG, BMP, and others. The library provides a forward API to load image data into memory, making it easy to work with image files in C.
4. Version Control: ○ Git: This version control system helps keep track of code changes, making it easy to collaborate or manage different versions of project. Git can be used alongside platforms like GitHub for code hosting and sharing.
5. Other Tool: Command Line Interface (CLI): The program is designed to be run from the command line, where the user provides the path to the image file as an argument. This simplifies the interaction and makes the system easy to use in environments

where GUI tools are not available.

- Terminal (Windows Command Prompt or Linux Shell): The terminal is used to execute the program and provide the image path.

System Workflow Summary:

- User Interaction: The user provides the path to the image via the command-line argument, which is passed to the system.
- Image Loading: The program loads the image using `stb_image`.
- Image Processing: The system processes the image by converting it to grayscale, calculating the luminance, and gamma-correcting the pixel values.
- ASCII Conversion: The system then converts the processed grayscale data into ASCII characters based on their intensity.
- Output Generation: The ASCII representation is saved to a `.txt` file, which the user can open and view.

This architecture offers a simple yet efficient way to convert images into ASCII art via a command-line interface, using powerful libraries like `stb_image` for image loading and processing. The modularity of the design allows for easy future expansions, such as adding a GUI or improving error handling and performance optimizations.

## 4.2 METHODOLOGY

**This section outlines the methodology used in the Image-to-ASCII Art Conversion Project.** The goal of the project is to transform a digital image into an ASCII art representation, a text-based representation that mirrors the original image

using ASCII characters. Below is a detailed explanation of the work carried out, including the techniques, procedures, and tools used in the project.

### 3.1 Image Loading and Preprocessing

The first step in the methodology involves loading the image that needs to be converted into ASCII art. For this task, the `stb_image` library is utilized, which is a single-file library written in C for loading images. The library supports various image formats like PNG, JPEG, and BMP, which makes it versatile for handling different types of input images. Once the image is loaded, its dimensions (width, height) and color channels (RGB or RGBA) are extracted. The image is then processed pixel by pixel. Since the goal is to convert the image to grayscale, each pixel's red, green, and blue (RGB) values are accessed individually.

Procedure:

1. Load the image using `stbi_load()` from the `stb_image` library.
2. Extract the pixel data, width, height, and number of color channels.
3. Handle any errors in loading the image, ensuring the program terminates gracefully in case of failure.

#### 4.1.1 Grayscale Conversion and Linearization

Once the image is loaded, the next step is to convert the color image (RGB) into grayscale, which simplifies the image to a single intensity value. To do this, each pixel's RGB values are linearized. The conversion from sRGB (standard RGB) to linear RGB is carried out based on the gamma correction formula, which adjusts for non-linear color response

## 5. RESULTS AND ANALYSIS

The image-to-ASCII conversion project successfully demonstrated the ability to transform digital images into text-based representations using ASCII characters. The

program was tested with various input images to evaluate its performance in terms of accuracy, visual fidelity, and computational efficiency.

## 1. Grayscale Conversion

The program effectively converted RGB images to grayscale by applying gamma correction and luminance calculations based on ITU-R BT.709 standards. This step ensured that the intensity information was accurately preserved, forming the basis for ASCII mapping

## 2. ASCII Mapping

The grayscale values were mapped to ASCII characters based on their perceived intensity, using a predefined character set ranging from dark to light (e.g., @, %, #, \*, =, -, ., and space). The mapping algorithm ensured that darker regions of the image were represented by denser characters, while lighter regions used sparse characters.

## 3. Output Quality

The generated ASCII art successfully captured the essence of the original images, with recognizable patterns and shapes. However, certain limitations were observed:

- **Resolution Trade-off:** Reducing the image size for ASCII conversion led to a loss of fine details, which is inherent to this method

## 4. Performance Evaluation

The program demonstrated efficient processing times for small-to-medium-sized images (up to 500x500 pixels). Memory usage was optimized through dynamic allocation for grayscale and ASCII arrays. However, performance could be further improved for larger images by parallelizing pixel processing.

## 5. Comparison with Existing Tools

When compared with other tools like `img2xterm` and `jp2a`, our program achieved comparable results in terms of grayscale accuracy but lacked advanced features like color ASCII art or half-block character support. These features could be explored in future iterations.

## 6. Key Observations

- The quality of ASCII art depends heavily on the input image's resolution and contrast.
- Using a more diverse character set or incorporating colored ASCII could enhance the visual appeal.
- Real-time performance is achievable with further optimizations.

**Character Set Limitations:** The fixed set of ASCII characters sometimes resulted in less precise shading compared to tools using extended character sets or color information

## 4. FUTURE ENHANCEMENT


The image-to-ASCII converter provides a functional foundation for text-based image representation, but several enhancements could elevate its utility and user experience. Below are potential improvements for future development:

### 1. Implementation of Color ASCII Art:

Incorporate ANSI escape codes to map RGB values to colored text characters, enabling vibrant and visually distinct ASCII outputs.

**2. Addition of Customizable Character Sets:**

Allow users to select from predefined ASCII character sets (e.g.,

@%#\*+=- : . or ) or define custom mappings for personalized results.

**3. Integration of Batch Processing:**

Enable bulk conversion of multiple images in a single execution to streamline workflows for large datasets.

**4. Development of a Basic Graphical Interface:**

Replace the command-line interface with a simple GUI featuring file selection menus, sliders for brightness/contrast adjustments, and real-time previews.

**5. Enhanced Error Handling and Feedback:**

Improve error messages for unsupported image formats and invalid inputs, providing actionable guidance to users.

**6. Dynamic Resolution Scaling:**

Add options to resize ASCII output independently of the input image dimensions, balancing detail preservation and file size.

**7. Cross-Platform Portability:**

Adapt the tool for web or mobile platforms using lightweight frameworks to broaden accessibility.

## CONCLUSION

This project successfully demonstrates the implementation of a versatile and robust tool capable of converting digital images into ASCII art through advanced pixel processing and color mapping techniques. From handling grayscale conversion to optimizing ASCII character mapping, the program provides a comprehensive solution for transforming visual data into text-based representations. The project emphasizes the importance of efficient pixel manipulation, color space transformations, and output formatting in achieving high-quality ASCII art.

**Key achievements** include:

- **Comprehensive Functionality:** Conversion of RGB images to grayscale using gamma-corrected luminance calculations.
- **Modular Design:** Separation of image loading, pixel processing, and ASCII mapping into distinct components.
- **Error Handling:** Robust checks for invalid image formats and memory allocation failures.
- **User-Friendly Interface:** Command-line execution with clear input/output instructions.

Overall, the image-to-ASCII project underscores the power of software in bridging visual and textual data representations. The lessons learned during development—such as the importance of color space accuracy, pixel-level precision, and memory efficiency—will serve as a foundation for future projects in image processing and computational art.

The project not only achieves its objectives but also opens avenues for enhancing ASCII art quality through advanced techniques like dithering or machine learning-based character mapping. It is a valuable tool for anyone seeking to explore the intersection of digital images and text-based art.



## REFERENCES

- [1] Wikipedia. (n.d.). *Grayscale*. Retrieved March 2, 2025, from <https://en.wikipedia.org/wiki/Grayscale>
- [2] Solarian Programmer. (2019, June 10). *C Programming - Reading and writing images with the stb\_image libraries*. Retrieved from [https://solarianprogrammer.com/2019/06/10/c-programming-reading-writing-images-stb\\_image\\_libraries/](https://solarianprogrammer.com/2019/06/10/c-programming-reading-writing-images-stb_image_libraries/)
- [3] Barrett, S. (n.d.). *stb\_image.h - v2.27 - public domain image loader*. Retrieved March 2, 2025, from [https://github.com/nothings/stb/blob/master/stb\\_image.h](https://github.com/nothings/stb/blob/master/stb_image.h)
- [4] Wikipedia. (2025, March 12). *Pixel*. Retrieved March 2, 2025, from <https://en.wikipedia.org/wiki/Pixel>
- [5] ImageJ Wiki. (n.d.). *Color Image Processing*. Retrieved March 2, 2025, from <https://imagej.net/imaging/color-image-processing>