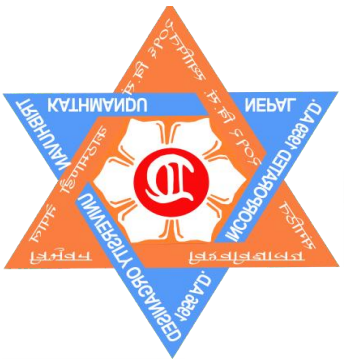

TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

Proposal

On

Image to ASCII Art

Submitted By:

[Diwen Baniya] [THA081BCT010]

[Kushal Joshi] [THA081BCT015]

[Parth KC] [THA081BCT019]

Submitted To:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

February , 2025

ABSTRACT

ASCII Art is a way for generating artwork with text characters. This project uses C to transform digital images into ASCII representations. The goal is to explore the artistic side of programming while learning picture manipulation in C.

The project converts photos into distinct creative representations, improves comprehension of image processing in C. It is a fun and engaging approach to creating text-based images.

Using the stb_image library, this project takes an image from disk, analyzes its pixel data, and translates it to ASCII characters in order to preserve its visual essence. The final result is stored as a.txt file.

TABLE OF CONTENT

TABLE OF FIGURES

LIST OF ABBREVIATIONS

1. stb: Set Top Box
2. jpeg: Joint Photographic Experts Group
3. png: Portable Network Graphics
4. ASCII: American Standard Code for Information Interchange

INTRODUCTION

1.1 Background Introduction

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:

- **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 C programming language to create a program to convert image to its equivalent ASCII art using stb_image library.

1.2 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.

1.3 Problem Definition

The main objective of this project is to read a image from the disc and convert it into ASCII art and write that image in the disc as a .txt file. We hope to achieve this by using stb_library.

1.4 Objectives

Our aim is to create an art creation program that can transform preexisting image to ASCII art that conserve the feel of original image. We hope to create a program that is fun to use and can produce results that can be appreciated by the user

Throughout this project, we intend to accomplish the following goals:

- This program should be able to read the image in the disk and be able to manipulate that image to achieve its function.
- We hope that this program can create ASCII art reminiscent to the image it reads.
- program can save the result without difficulty.
- Generate the ASII art considering the properties of original image.

By doing this project we were able to learn the basic of handling and manipulating various image files

1.5 Scope and Applications

This project focus on application of stb_image library on manipulating image files. By doing this project we were able to learn to manipulate .jpeg and .png files in various ways. A novice programmer will be able to apply these concepts in further working with images in program. As we will require to frequently work with images while programming the concepts learned while doing this project will be extremely valuable.

2. LITERATURE REVIEW

The literature review of this project will cover the following topics:

• Various types of image files I.e:, jpeg, .png

• The history and development of the C programming language.

2.1 Various types of images files

An **image file format** is a file format for a digital image. There are many formats that can be used, such as JPEG, PNG, and GIF. Most formats up until 2022 were for storing 2D images, not 3D ones. The data stored in an image file format may be compressed or uncompressed.
JPEG

JPEG(Joint Photographic Expert Group Image) is a image file type with extensions. “.jpg” , “.jpeg” . it is currently the most widely used lossy compression format for still images. It's particularly useful for photographs; applying lossy compression to content requiring sharpness, like diagrams or charts, can produce unsatisfactory results. JPEG is actually a data format for compressed photos, rather than a file type.

PNG

The PNG (pronounced "**ping**") image format uses lossless compression, while supporting higher color depths than GIF and being more efficient, as well as featuring full alpha transparency support.

PNG is widely supported, with all major browsers offering full support for its features.

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.

1972 C	C was first released.
1978 K&R	The first edition of “The C Programming Language” book by Bria Kernighan and Denn Ritchie, often referred to as K&R C. It served as reference until a formal standard w established.
1989 C-89/ ANSI	ANSI standardized the language (ISO/IEC 9899:1990), introducing standard libraries a features like function prototypes, void pointers etc.
C	
1990 C-90	Minor updates and fixes on C89.
1999 C-99	Support for variable length arrays, new data types, and inli functions (ISO/IEC 9899:1999
2011 C-11	Introduction of additional data types, multi-threading, and improve Unicode (ISO/IE 9899:2011).
2018 C-18	Minor updates and fixes on C11 (ISO/IEC 9899:2018).

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),

embedded systems, and performance-critical applications. By balancing abstraction 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.

PROPOSED 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.

System Architecture

The system architecture can be broken down into the following components:

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 the image 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).

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:

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

2. 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.

3. After linearization, the luminance of each pixel is calculated using the formula:
   $\text{Luminance} = 0.2126 \cdot R + 0.7152 \cdot G + 0.0722 \cdot B$ $\text{Luminance} = 0.2126 \cdot R + 0.7152 \cdot G + 0.0722 \cdot B$
   This step is crucial because the luminance value represents the perceived brightness of a pixel, which is essential for the ASCII conversion.

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

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:

1. 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).

2. 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.

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

Output Module
Function: The final ASCII representation of the image is output as a text file.
How It Works:

1. 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.

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

Error Handling and Validation
Function: The system includes basic error handling to ensure smooth execution.
How It Works:

1. The program checks if the input image file is valid and can be opened.

2. If the image cannot be loaded or the file is invalid, the system will output an error message and terminate gracefully.

3. 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
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.

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 is lightweight and easy to configure.

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 straightforward API to load image data into memory, making it easy to work with image files in C.

- **stb_image_write**: While not used directly in the project for output generation (since you're generating a `.txt` file), it is often used for saving images if needed in future expansions or for testing purposes.

Version Control: Git: This version control system helps keep track of code changes, making it easy to collaborate or manage different versions of the project. Git can be used alongside platforms like GitHub for code hosting and sharing.

Other Tools:

- **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.

3.

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. 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.

4.1 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.

Algorithm for RGB to Grayscale Conversion:

1. **Obtain RGB Values**: For each pixel, retrieve its RGB components.

2. **sRGB to Linear RGB Conversion**:

   Apply gamma correction using the following formula:

$$\text{Linear RGB} = \begin{cases} \frac{\text{RGB}}{12.92}, & \text{if RGB} \leq 0.04045 \\ \left(\frac{\text{RGB}+0.055}{1.055}\right)^{2.4}, & \text{otherwise} \end{cases}$$

3. **Luminance Calculation**:

   Compute grayscale intensity using weighted RGB values:

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

4. **Gamma Compression**:

   Apply inverse gamma correction to the luminance value to convert it back to sRGB space, yielding the final grayscale value.

5.

4.1.2 ASCII Mapping

After obtaining the grayscale values for each pixel, the next step is to map these values to ASCII characters. The grayscale values range from 0 to 255, and each value corresponds to a specific ASCII character. The characters are chosen based on their visual intensity, with darker characters representing lower grayscale values (darker pixels) and lighter characters representing higher values.

The character set is designed from dark to light, using characters like @, #, *, +, =, -, :, and .. This ensures that the ASCII output has a good balance of visual representation.

Procedure for ASCII Mapping:

1. For each pixel in the grayscale image, obtain the grayscale intensity.

2. Map the grayscale intensity to an ASCII character based on its value. The darker pixels are mapped to denser characters, and lighter pixels are mapped to characters that are less visually dense.
3. Store the resulting ASCII characters into an array, representing the final ASCII art output.

4.2 Image Output

Once the ASCII art has been generated, it is stored in a text file. The output consists of rows of characters that correspond to the rows of pixels in the original image. A file is created where each line of the file contains one row of ASCII characters, representing the image in text form.

Procedure:

1. Open a file (e.g., `ascii_art.txt`) for writing.
2. Write each line of ASCII characters to the file, ensuring that each row of the original image corresponds to a line of text.
3. Close the file once all data has been written.

4.2.1 Error Handling and Performance Considerations

During the image loading, processing, and output phases, error handling is critical to ensure the program behaves reliably under all conditions. This includes:

- Verifying that the image was loaded correctly.
- Checking the image dimensions to ensure the output is within a reasonable size.
- Handling file I/O errors when saving the ASCII art.

Additionally, performance optimizations can be considered, particularly for large images. Techniques such as multi-threading or optimized memory usage could be applied to speed up the process for high-resolution images.

4.2.2 Testing and Debugging

After completing the main functionality of the program, extensive testing is conducted. This includes:

- Testing with images of various sizes and formats to ensure robustness.
- Verifying that the grayscale conversion and ASCII mapping yield accurate results.
- Handling edge cases such as very dark or very light images to ensure that the output still resembles the original image in terms of contrast and structure.

Debugging is carried out using print statements and breakpoints to track down any errors or performance bottlenecks. Unit tests and visual verification of output are used to ensure that the program functions as expected.

4.3 Future Extensions

Future work on this project may include developing a **Graphical User Interface (GUI)** to allow users to select images more easily and customize their ASCII art conversion preferences. Additionally, performance improvements such as optimizing the algorithm for large images or supporting more complex image formats could be considered. Further research on advanced image processing techniques may help improve the overall quality of the ASCII art output.

5. SCOPE AND APPLICATIONS

Scope

This project focuses on converting digital images into ASCII art—a text-based representation that mimics the visual content of an image using characters. The key processes involved include:

- **Image Loading**: Supporting various image formats to be processed.
- **Color Space Conversion**: Transforming images from their native color spaces (such as sRGB) into grayscale, which simplifies the mapping of pixel intensities to ASCII characters.
- **ASCII Mapping**: Assigning ASCII characters based on the grayscale value of each pixel or region, creating a character-based representation of the original image.

Applications

1. **Artistic Expression**:
    - Offers artists and designers a unique medium for reinterpreting digital images as ASCII art, combining technology with creativity.
    - Allows users to generate visually captivating works using only text characters.
2. **Text-Based Interfaces**:
    - Enables image display in environments where graphical support is unavailable, such as terminals or text editors.
    - Useful for systems that have limited graphical rendering capabilities.
3. **Accessibility Tools**:
    - Can be integrated into accessibility tools to provide text-based image descriptions that can be read aloud by screen readers, aiding visually impaired users.
4. **Educational Purposes**:
    - Serves as an interactive tool to teach concepts related to image processing, grayscale conversion, and character mapping in programming.
5. **Nostalgia and Entertainment**:
    - Appeals to fans of retro computing aesthetics and those who enjoy creative reinterpretations of digital media in a classic, text-only format.
6. **GUI Development (Future Extension)**:
    - Developing a graphical user interface (GUI) could enhance user experience by making the image selection and conversion process more intuitive and interactive.
7. **Performance Comparison (Future Extension)**:
    - The project could be extended to compare conversion speeds across different programming languages or execution methods, exploring performance optimizations.

This project blends creativity and technical expertise in image processing, offering diverse applications in art, education, accessibility, and entertainment, with potential for further expansion into user-friendly interfaces and performance optimization studies.

6. TIME ESTIMATION

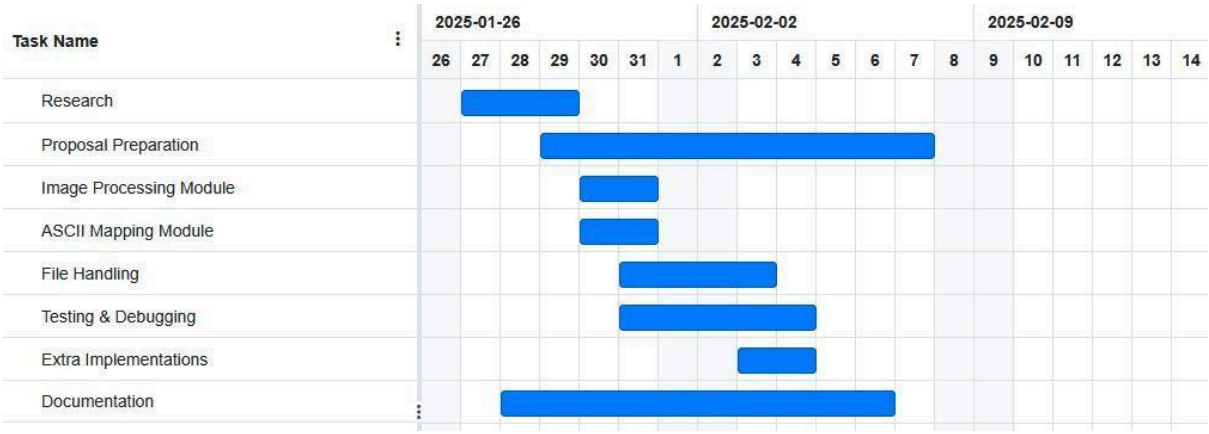

Fig. Time estimation chart

7. FEASIBILITY ANALYSIS

Practical Use

The **Image-to-ASCII Art Conversion** project can be practically applied in several areas, including artistic expression, retro computing, and accessibility tools. It allows users to transform images into text-based art that can be displayed in text environments like terminals, websites, or even as part of email signatures. The project is especially relevant in resource-constrained environments where graphical rendering is either limited or unnecessary. ASCII art is also popular in retro and indie design, making the tool useful for designers and developers seeking a unique aesthetic.

Cost

The cost of developing and running this project is minimal:

- **Development Costs**: Since the project relies on freely available libraries like stb_image and standard development tools, there are no direct financial costs involved in its creation. The main investment is the time required for development, testing, and debugging.
- **Hardware Requirements**: The project can run on any standard computing system without specialized hardware, making it accessible for most developers and users.
- **Operational Costs**: Once developed, the tool can be used without significant ongoing costs. It could be hosted locally or distributed as a lightweight application.

Complexity

The overall complexity of this project is moderate:

- **Technical Complexity**:
    - The key challenges lie in image processing, converting pixel intensities to grayscale, and mapping them to ASCII characters.
    - The project uses basic image processing techniques like grayscale conversion and character mapping, which are straightforward with modern libraries.
    - However, future extensions (such as adding a graphical user interface or optimizing for performance) might introduce additional complexity.
- **Usability Complexity**:
    - The current command-line-based implementation is simple to use for those familiar with programming environments but could be challenging for non-technical users.
    - Developing a GUI would increase its accessibility and user-friendliness, but also add to development complexity.

Scalability

The project is highly scalable for small to moderate image sizes. However, converting very large images could slow down the conversion process due to the nature of pixel-by-pixel analysis and ASCII mapping. Optimization techniques could be employed if performance issues arise when processing large datasets.

Limitations

- **Visual Limitations**: The output is constrained by the limitations of ASCII characters in terms of how well they can represent the fine details of complex images. This makes it better suited for images with strong contrasts and simple designs.
- **Color and Detail Loss**: The grayscale and text conversion process simplifies the image, meaning that fine color details and complex patterns will be lost.

References

[1] Computersciencecafe.com, "Image Representation Techniques," [Online]. Available: https://www.computersciencecafe.com/15-image-representation.html. [Accessed: Feb. 2025].

[2] Solarianprogrammer.com, "C Programming: Reading and Writing Images with stb_image Libraries," [Online]. Available: https://solarianprogrammer.com/2019/06/10/c-programming-reading-writing-images-stb_image-libraries/. [Accessed: Feb. 2025].

[3] Wikipedia, "Grayscale," [Online]. Available: https://en.wikipedia.org/wiki/Grayscale. [Accessed: Feb. 2025].