IMAGE PROCESSING SOULTION

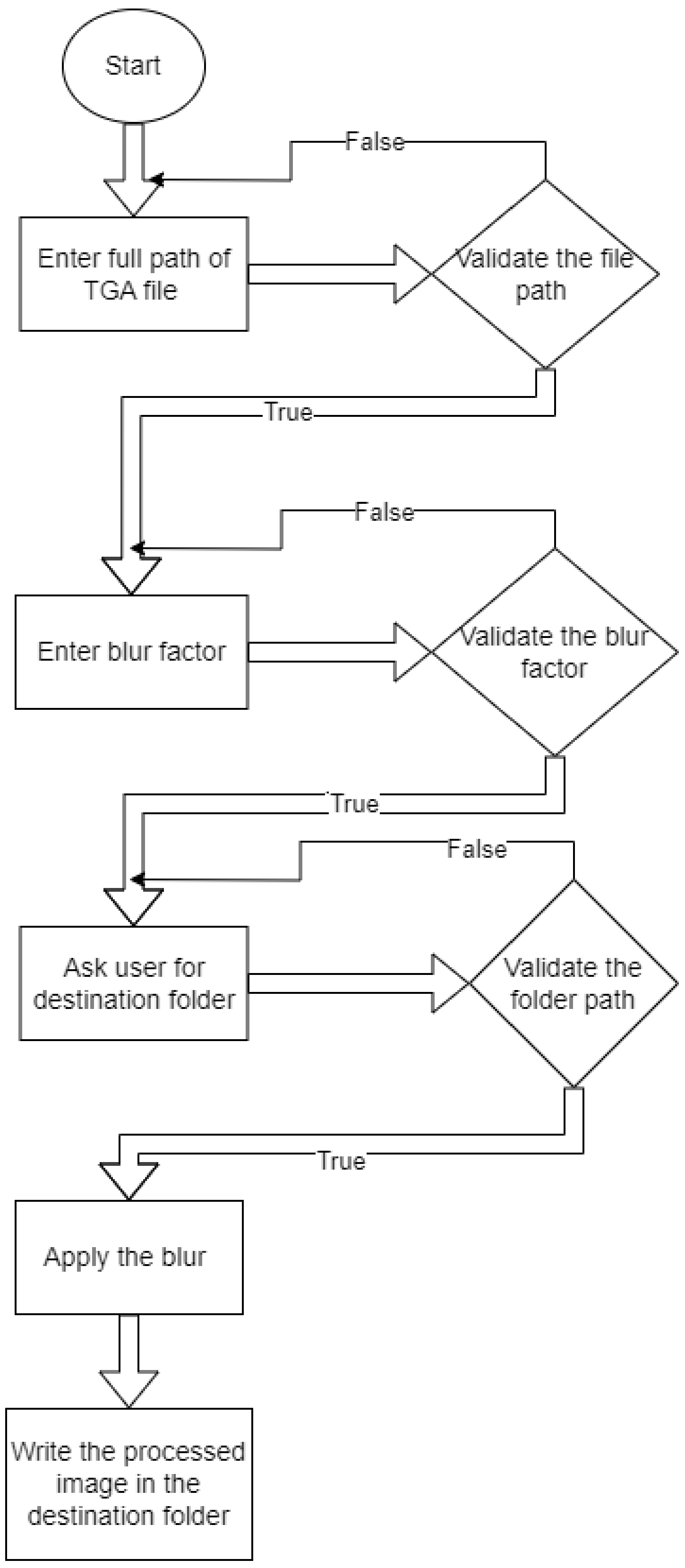
BASIC OVERVIEW

Upon application launch, users are presented with a menu of options, accessible by selecting the corresponding number. To ensure the application's extensibility and maintainability, the Model-View-Controller (MVC) design pattern is employed. The Model handles the core business logic, such as applying the blur filter, while the View manages user interactions and information display. The Controller acts as the intermediary between the Model and View, forwarding user input to the Model for processing. In alignment with SOLID principles, the "FileOperations" class is introduced to centralize file-related operations, including existence checks, validity assessments, and file name retrieval. This structured approach enhances code organization and facilitates future enhancements

DESCRIPTION OF BLUR OPERATION

When selecting the "Apply blur filter" option, the program meticulously guides the user through the process. It initially requests the complete TGA file path, followed by the user-specified blur factor within the range of 0.0 to 1.0, and the destination folder. To ensure a smooth operation, the program currently supports paths in quotes, aligning with Windows' "Copy as Path" convention.
A stringent validation process ensures that the user provides valid input for all options (TGA file, Blur factor, Destination folder) before proceeding. For now, I am supporting 24 and 32 bit, uncompressed true-color image (Image type -2).
Once user input is validated, the program engages the TGAFileOperation class for reading and writing TGA files. This class, which inherits from FileOperations, provides a foundation for future extensibility by allowing the addition of new file format support with minimal effort.
The data read from the file is then passed to the Model, which processes the information and applies the blur filter using the ApplyBoxBlurFilter function. Following the filter application, the TGAFileOperation class generates the new blurred image at the specified location.
The name of the blurred image follows a naming convention: "Blur_OriginalImageName," with "Blur_" as the prefix. In the case of name conflicts, the program appends a numerical suffix to ensure each output file has a unique name.
Throughout this process, the CLI interface provides detailed information, distinguishing between errors and instructions using distinct prefixes like “[ERROR]->” for error and “-->” for instruction.
Additionally, a Log class records application progress, errors, and warnings in a text file, which is defined by the programmer in the main function and located alongside the executable. This dual output system enhances user guidance and provides valuable insights for programmers tracking application behaviour.

WORK FLOW


Start
Enter full path of TGA file
Validate the file path
False
True
Enter blur factor
Validate the blur factor
False
True
Ask user for destination folder
Validate the folder path
False
True
Apply the blur
Write the processed image in the destination folder

CLASS OVERVIEW

1. ImageProcessingController (Main Class): The concrete controller class, assumes full responsibility for orchestrating the entire application. Its constructor diligently initializes critical objects from classes including View::ImageProcessingView, Model::ImageProcessingModel, Log::LogSystem, and FileOperations::FileOperations. The StartApplication function serves as the program's driving force, and it gracefully concludes the operation when the user opts to exit. Notably, this class employs the UserInputType Enum to facilitate user input. Future expansions and feature additions are seamlessly accommodated by simply editing this Enum, underscoring the system's adaptability and extensibility.
2. IFileOperation (Abstract Class): IFileOperation stands as an abstract class for file operations, incorporating crucial pure virtual functions like GetDirectoryPath, GetFilenameWithExtension, and IsValidFolderPath.
3. FileOperations: This primary class for file operations provides implementations for all the pure virtual functions defined in IFileOperation.
4. TGAFileOperation: TGAFileOperation is tasked with reading and writing TGA files, facilitating file operations specific to this file format.
5. ImageProcessingModel: As the central model class, ImageProcessingModel is responsible for implementing the business logic needed to apply a blur filter.
6. IView (Abstract Class): View acts as the abstract class for the view, with important pure virtual functions for printing on the screen and obtaining paths from the user.
7. ImageProcessingView: The primary view class, ImageProcessingView, implements logic for functions like GetFormattedFilePath and PrintOnScreen.
8. LogSystem: LogSystem manages the application's logging system. It is initialized with the log file name at the application's outset, located in the same directory as the executable. This object is passed to various classes, allowing them to log messages with different log levels. Upon termination, the destructor of this class ensures the closure of the log file with a closing comment.

BOX BLUR ALGORITHM

The chosen algorithm for this application is the Box Blur algorithm, effectively processing 24-bit and 32-bit TGA images. For 24-bit image each element within the "m_OriginalTGAImage" vector represents the RGB (Red, Green, Blue) colour values for an individual pixel and for 32-bit images, the "m_OriginalTGAImage" vector accommodates RGBA (Red, Green, Blue, Alpha) values.
The core logic involves iterating through each pixel, starting from the origin and progressing to the last pixel. For each pixel, the algorithm considers neighbouring pixels, validating their values. It calculates the sums of the R, G, and B values of these neighbouring pixels (RGBA for 32-bit image). The extent of iteration over neighbouring pixels depends on the specified blur factor; a higher blur factor entails more extensive iteration, including the aggregation of RGB values (RGBA for 32-bit image).
Subsequently, the algorithm locates the corresponding pixel in the "m_OriginalTGAImage" vector and updates its RGB/RGBA value with the average of the R, G, and B values (RGBA for 32-bit image). This average is calculated by summing the individual values and dividing by the number of elements involved in the calculation.
Explanation of each step:
1. Calculate Blur Size: The process commences by computing the blur size, a pivotal parameter derived from the provided `blurFactor`. This size governs the extent of the pixel neighbourhood considered during the blur operation.
2. Determine Channel Count: To ensure compatibility with both 24-bit and 32-bit images, the channel count is determined. This count serves as a foundational element for subsequent calculations, such as `AlphaSum` and pixel value updates.
3. Iterate Through Pixels: Employing nested loops, the algorithm systematically traverses every pixel within the image.

4. Initialize Accumulators: Accumulators, responsible for tracking colour channels (red, green, blue and alpha), and a pixel count (`NumPixels`) are initialized to zero. These accumulators play a crucial role in determining the average colour values within the pixel neighbourhood.
5. Neighbourhood Exploration: The function employs further nested loops to explore the neighbourhood of pixels surrounding the current pixel. The neighbourhood's size is dynamically determined by `BlurSize`.
6. Coordinate Calculation: For each pixel within the neighbourhood, the X and Y coordinates of the neighbouring pixel are computed (`PixelX` and `PixelY`).
7. Boundary Verification: A crucial boundary check ensures that the neighbouring pixel falls within the image's boundaries, preventing out-of-bounds calculations.
8. Index Computation: When a neighbouring pixel resides within the boundaries, the function calculates its index within the "m_OriginalTGAImage" vector based on its coordinates. This index is instrumental in accessing the neighbouring pixel's colour channel values.
9. Colour Channel Accumulation: The colour channel values, including red, green, and blue, are systematically accumulated within the designated accumulators (`RedSum`, `GreenSum`, and `BlueSum`). In the case of processing 32-bit images, `AlphaSum` is also computed.
10. Count Increment: The count of pixels within the neighbourhood (`NumPixels`) is incremented for each valid neighbouring pixel.
11. New Colour Value Calculation: Following the comprehensive iteration through the neighbourhood, the function calculates the updated colour values for the current pixel. These newly averaged colour values are then assigned to the current pixel's position within the "m_OriginalTGAImage" vector, ensuring that the central pixel effectively reflects the average colour values of its neighbourhood.

LOGGING SYSTEM

I have implemented a comprehensive log system to monitor and record the application's progress. This log system is designed with three distinct log levels: "Error," "Info," and "Warning." Upon the application's initiation, a log class object is instantiated, specifying the log file's name. This log object is then efficiently shared across all application classes, facilitating the seamless utilization of the log system. It plays a pivotal role in tracking and documenting various aspects of the application's functionality, promoting enhanced transparency and diagnostic capabilities throughout the development process.

C++17 – STD::OPTIONAL AND STRUCTURAL BINDING

During the process of reading and writing TGA files, I encountered a distinct challenge that necessitated the concurrent return of both a Boolean value signifying the success of the operation and an associated error string in case of any issues. To address this dual requirement, I made the decision to utilize a pair consisting of an optional string data and a boolean variable and collect then in structural binding. This approach enables me to effectively handle error conditions by returning a false boolean value in conjunction with an optional string variable containing an error message. Subsequently, in the calling function, I can seamlessly examine the optional variable to determine the presence of an error message and, if required, display it to provide clarity and diagnostic information. This method ensures a robust and informative error-handling mechanism.

IMPORVMENTS IN UPCOMING VERSIONS

1. Advanced Blur Algorithm Exploration: A quest for enhanced image blurring techniques leads to the exploration of alternative algorithms. Gaussian blur, a prominent choice, relies on the Gaussian function to determine the transformation applied to each pixel in an image. This pursuit aims to harness more efficient and sophisticated blurring methods.

2. Optimized Log Management: Elevating log management involves implementing critical features such as log rotation. This functionality enables the control of log file size and the retention of a specified number of historical log files, adding practicality to real-world applications. To further enhance the system's robustness, thread safety measures are also considered, catering to programs utilizing multithreading for seamless and concurrent logging.
3. Extended File Format Support: In the current configuration, the application supports the TGA file format. However, with additional development time, the scope of supported file formats can be broadened to include popular options like PNG and JPG, enhancing the application's versatility and compatibility.

Supported Visual studio version – VS 2022