Documentation for Node-Based Image Processing Framework

Mixar

April 15, 2025

1 Abstract

This documentation provides an in-depth explanation of the Node-Based Image Processing Framework, which allows users to manipulate images through a variety of operations in a modular and flexible way. The system is designed using nodes that perform individual image processing tasks and can be connected together to form a complete pipeline.

2 Introduction

This system implements a node-based approach for image manipulation using a C++ framework. The architecture is inspired by visual programming environments where users interact with individual nodes that represent specific image processing operations. The modular design allows easy configuration, extension, and customization, making it highly adaptable for various image processing tasks.

3 System Architecture

3.1 Node-Based Design

The core design of the system revolves around **nodes**. Each node represents an individual image processing operation, such as applying a filter, adjusting brightness, or performing edge detection. These nodes can be connected to form a processing chain, where the output of one node serves as the input to the next.

3.2 NodeGraph Management

The `NodeGraph` class manages all nodes and their connections. It ensures that nodes are executed in the correct order and handles any dependencies between nodes. The graph structure allows for easy visualization and manipulation of the processing pipeline.

3.3 Modularity

Each image processing task is encapsulated in a separate class. New nodes can easily be added to the system to support additional operations, maintaining the flexibility of the architecture.

4 Key Components

4.1 Nodes

- **ImageInputNode**: This node is responsible for loading images from the disk and providing them to the graph. It supports multiple image formats such as JPG, PNG, and BMP.
- **OutputNode**: After the image processing pipeline is complete, the `OutputNode` saves the result to a file. It supports different formats and allows the user to specify the output location.
- **BrightnessContrastNode**: This node applies brightness and contrast adjustments to the input image. Users can control the degree of adjustment via sliders.
- **ColorChannelSplitterNode**: This node splits the input image into its individual color channels (RGB or RGBA), allowing for individual manipulation of each channel.
- **BlurNode**: This node implements Gaussian and directional blur effects. The user can adjust parameters like radius and angle for directional blur.
- **ThresholdNode**: Converts the input image to a binary image using various thresholding methods such as binary, adaptive, and Otsu's method.
- **EdgeDetectionNode**: Implements edge detection algorithms, such as Sobel and Canny, allowing the user to configure parameters like kernel size and thresholding.

4.2 Node Interactions

Nodes can be connected by linking their outputs to the inputs of other nodes. This creates a directed acyclic graph (DAG) structure where the image flows through a series of transformations, and the final result is generated at the end.

5 Implementation Details

5.1 Edge Detection Node

The `EdgeDetectionNode` provides two major edge detection algorithms: Sobel and Canny. The user can specify kernel size for Sobel and thresholds for Canny.

- **Sobel Edge Detection**: Uses the Sobel operator to calculate the gradient of the image and highlight edges.
- **Canny Edge Detection**: A more sophisticated edge detection algorithm, which involves multiple stages, including noise reduction, gradient calculation, non-maximum suppression, and edge tracking.

The parameters for both algorithms can be configured via the user interface, allowing for fine control over the edge detection process.

5.2 Threshold Node

The `ThresholdNode` converts the image into a binary image based on a specified threshold. The user can select from various thresholding methods:

- **Binary Thresholding**: Pixels above the threshold are set to one value, and pixels below it are set to another.
- **Adaptive Thresholding**: The threshold is dynamically adjusted based on local regions of the image.
- **Otsu's Thresholding**: Automatically determines an optimal threshold value based on the image histogram.

6 Third-Party Libraries

6.1 OpenCV

OpenCV is an open-source computer vision library that provides a comprehensive set of tools for image processing. It is used extensively in this system for tasks like edge detection, thresholding, and image manipulation. OpenCV is highly efficient and widely used in industry for computer vision and image processing applications.

6.2 ImGui

ImGui is a simple and fast graphical user interface (GUI) library used to create in-app UIs. In this system, ImGui is used for displaying sliders, buttons, and checkboxes that control the parameters of various nodes, like the blur radius or thresholding method.

7 Compilation and Setup

7.1 Dependencies

To build and run the project, the following libraries are required:

- **OpenCV**: For image processing tasks.
- **ImGui**: For rendering the user interface components.

7.2 Build Instructions

- **Dependencies**:
  - OpenCV
  - ImGui
- **Compilation**: The project is built using CMake.
  - Clone the repository.
  - Install OpenCV and ImGui.
  - Run `cmake ..` to generate build files.
  - Use `make` to compile the project.
- **Run**: After compilation, the application can be run from the terminal. The user can interact with the nodes and perform image manipulations.

8 User Instructions

8.1 Interaction with the System

Upon running the program, users are presented with a menu of actions:

- Load an image file.
- Apply various image processing operations such as blur, edge detection, and thresholding.
- Adjust image parameters like brightness and contrast.
- Merge or split color channels.

Users can enter their choice by selecting the corresponding number from the menu.

8.2 Edge Detection Example

To apply edge detection:

- Choose the edge detection algorithm (Sobel or Canny).
- Specify the necessary parameters (e.g., kernel size for Sobel, thresholds for Canny).
- View and save the edge-detected image.

9 Future Enhancements

9.1 GUI Integration

The current system uses a console interface. For better usability, a full-fledged graphical user interface (GUI) can be developed using Qt or another GUI framework. This will allow for easier interaction, such as drag-and-drop functionality for connecting nodes.

9.2 More Image Processing Nodes

Additional nodes can be created to extend the capabilities of the system. Possible nodes could include:

- Histogram equalization

- Image rotation and scaling
- Image blending

10 Conclusion

The Node-Based Image Manipulation Interface provides an intuitive and flexible way to perform complex image processing tasks. By leveraging a modular design with node-based operations, the system is extensible, maintainable, and easy to use. With the help of OpenCV and ImGui, the system offers a solid foundation for creating sophisticated image manipulation applications.