Project Report: Image Boundary Detection and Segmentation System

Group Members

- Zihe Wang, 12312216
- Haoheng Liu, 12310627

Introduction

This project implements an interactive image segmentation tool based on the "Intelligent Scissors" algorithm, allowing users to accurately extract object boundaries through simple mouse operations. The system combines advanced edge detection with optimal path finding to deliver precise segmentation results with minimal user input.

System Function List

Core Functionality

1. **Real-time Path Tracing**
    - Dynamically computes and displays optimal boundary paths from seed points to cursor position
    - Automatic edge snapping to significant image features
    - Multi-segment path recording for complex contours
2. **Interactive Segmentation**

- Visual feedback with color-coded paths and control points
- Undo/redo functionality for path correction
- Path completion with automatic region filling

3. **Output Options**
   - Save segmented images in multiple formats
   - Copy results to clipboard
   - Automatic cropping to region of interest

Advanced Features

- **Cursor Snap**
  - Real-time gradient analysis within cursor neighborhood
  - Dynamic position adjustment to align with edge features
  - Visual feedback through highlighted edge points
- **Path Cooling**
  - Stability monitoring through path coalescence tracking
  - Automatic seed point generation at stable segments
  - Adaptive path segmentation for long boundaries

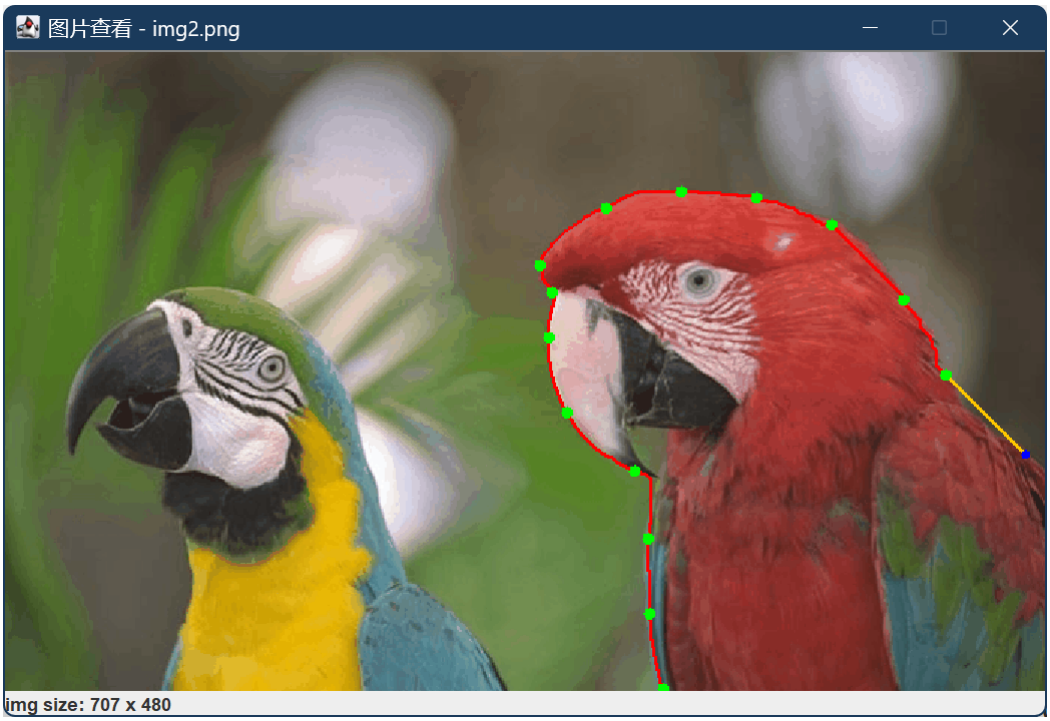
System Usage Instructions (with illustrations in pictures)

1.Image Selection

- Click "选择图片" button to open a file dialog
- Select an image file (JPG, PNG, GIF, or BMP)

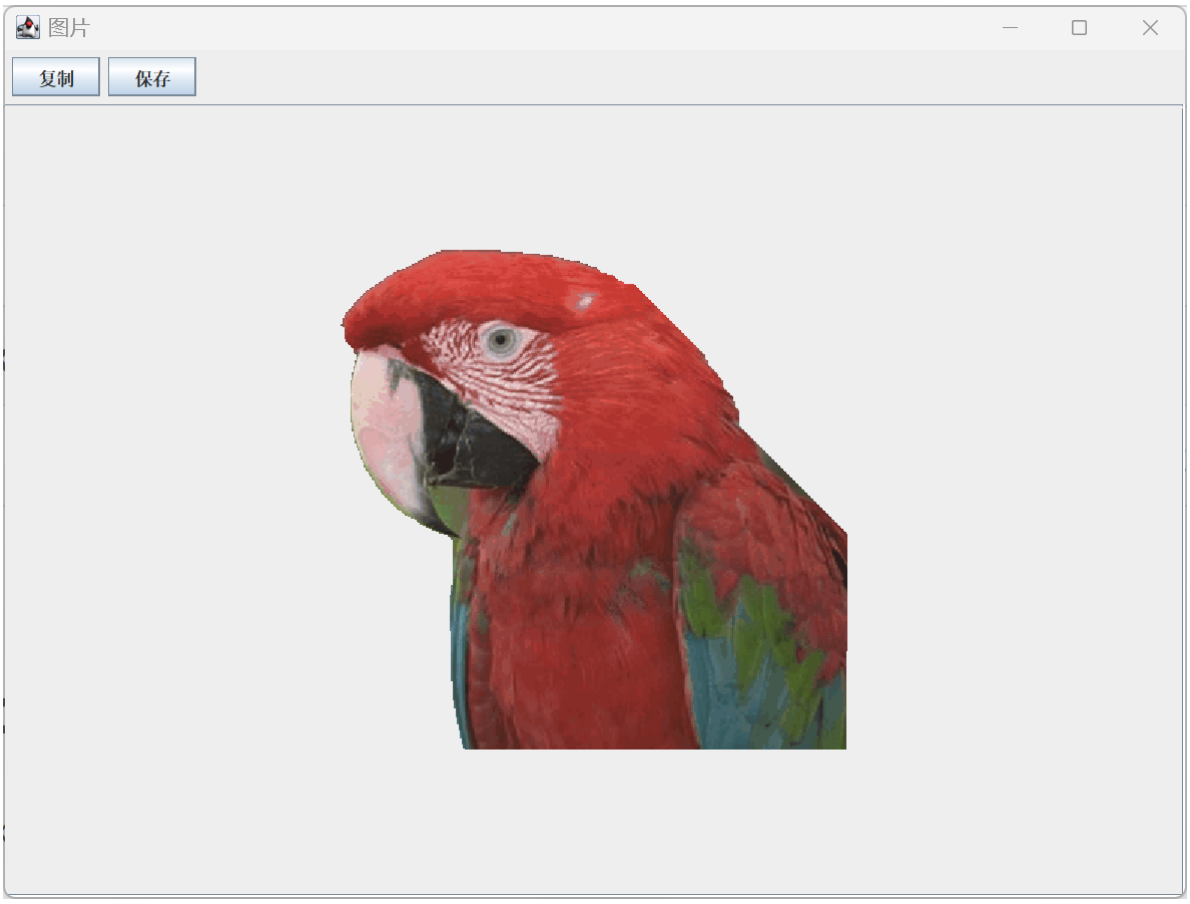
2.Boundary Tracing

Action	Functionality
Left Click	Set new path starting point or add control point
Left Double-Click	Complete path to first seed point and open save/copy dialog
Right Click	Remove most recently added control point
Right Double-Click	Reset all paths and tracking state
Mouse Movement	Live update of optimal path from last seed point to cursor position



3.Result Handling

- After double-clicking, a new window will show the segmented image
- Use "复制" button to copy the image to clipboard
- Use "保存" button to save the image to a file



4.Reset

- Double right-click to clear all paths and start over

System Structure Description

1. Class Overview

DijkstraAlgorithm.java

This class implements Dijkstra's algorithm for finding the shortest path between two points in a grid with weighted edges.

Key functions:

- `findShortestPath()`: Main method that implements Dijkstra's algorithm to find the shortest path between start and end points using a cost matrix
- `reconstructPath()`: Helper method that backtracks from the end point to reconstruct the path using the predecessor matrix
- `Node` inner class: Represents a grid point with its current distance, used in the priority queue

GrayConverter.java

This class handles conversion of color images to grayscale matrices.

Key functions:

- `convertToGrayMatrix()`: Converts a BufferedImage to a grayscale matrix using luminance calculation
- `calculateLuminance()`: Computes the perceived brightness of a color using standard luminance coefficients

Main3.java

This is the main application class that provides the GUI for image selection, path drawing, and region extraction.

Key functions:

- UI creation methods (`createSelectionWindow()`, `showImageWindow()`, `showScalableImage()`)
- Image processing methods (`selectImage()`, `cropToMinimumBounds()`)
- Clipboard operations (`copyImageToClipboard()`, `TransferableImage` inner class)
- File operations (`saveImageToFile()`, `getFormatName()`)
- `ImagePanel` inner class: Handles all mouse interactions and path drawing
    - Mouse event handlers for path creation and modification
    - Path drawing and visualization methods
    - Path management utilities (`updateCurrentPath()`, `distBetweenPoints()`)

SobelConverter.java

This class applies Sobel edge detection to convert grayscale images to cost matrices.

Key functions:

- `convertToCostMatrix()`: Main method that applies Sobel operators and converts to normalized cost matrix
- `calculateConvolution()`: Helper method that performs the convolution operation with a given kernel
- `printMatrix()`: Debug utility for matrix visualization

StuffAlgorithm.java

This class implements region filling and boundary detection algorithms.

Key functions:

- `surround()`: Main method that coordinates the three-step marking process
- `markOutside()`: Uses flood fill to mark all areas outside the drawn path
- `markBoundary()`: Identifies path points adjacent to outside areas
- `markRemaining()`: Marks all remaining interior points

2. Core Algorithms

Path Finding (Dijkstra's Algorithm)

The system uses an 8-directional variant of Dijkstra's algorithm for path finding:

1. Initializes all distances to infinity except the start point
2. Uses a priority queue to always expand the closest unvisited point
3. For each point, examines all 8 neighbors and updates their distances if a shorter path is found
4. Terminates when the end point is reached or queue is empty
5. Reconstructs path by backtracking from end point using predecessor matrix

Edge Detection (Sobel Operator)

The cost matrix generation uses Sobel edge detection:

1. Applies Sobel X and Y kernels separately to compute horizontal and vertical gradients
2. Combines gradients to compute magnitude at each point
3. Normalizes magnitudes to create cost values between 0 and 1
4. Inverts values so edges have low cost and smooth areas have high cost

Region Filling and Boundary Detection

The region processing algorithm has three phases:

1. **Flood Fill from Edges**: Marks all areas connected to image edges as outside (-1)
2. **Boundary Marking**: Identifies path points adjacent to outside areas as boundary (2)
3. **Interior Filling**: Marks remaining unmarked areas as interior (3)

3. Data Structures

Primary Data Structures

- **double[][] costMatrix**: 2D array storing movement costs between pixels (Sobel output)
- **int[][] boolMatrix**: 2D array marking different regions (path, boundary, interior, exterior)
- **List**: Used extensively to store paths and seed points
- **PriorityQueue**: Used in Dijkstra's algorithm to efficiently retrieve next closest point

Supporting Data Structures

- **BufferedImage**: The fundamental image representation throughout the system
- **Point**: Basic 2D coordinate representation
- **Queue<int[]>**: Used in flood fill algorithm for BFS traversal
- **Set**: Used to store unique points during region extraction

GUI-related Structures

- **JFrame**: Main application windows
- **JPanel**: Drawing surfaces and UI containers
- **Graphics2D**: Used for all drawing operations

- **MouseEvent**: Handles user interaction data

Project Summary

Key Features and Innovations

1. **Precise Boundary Detection**
    - Combines Sobel edge detection with Dijkstra's algorithm
    - Automatically follows strong edges in the image
    - 8-directional movement provides natural path shapes
2. **Interactive Workflow**
    - Real-time path updates as mouse moves
    - Visual feedback with colored paths and points
    - Undo functionality for error correction
3. **Automatic Optimization**
    - Path cooling prevents excessively long segments
    - Dynamic seed point placement maintains accuracy
    - Efficient region analysis for quick segmentation
4. **User-Friendly Output**
    - Automatic cropping to region of interest
    - Multiple output options (file, clipboard)
    - Scalable result viewing

Performance Considerations

1. **Efficient Path Finding**
    - Priority queue ensures optimal node selection
    - Early termination when reaching target
    - Limited search space through dynamic updates
2. **Memory Management**
    - Reuses buffers where possible
    - Only essential data retained between operations
    - Efficient image copying for display
3. **Responsive UI**
    - Separate thread for intensive computations
    - Progressive path updates
    - Minimal repaints through buffered drawing