Intelligent Scissors Implementation Report

CS563-463 Computer Vision Assignment 3 | Fall 2025

---

1. Identification

Instructor: Dr. Elmehdi Aitnouri | **Due:** November 10, 2025
Team: ReSpawn | **Members:** Masoud Rafiee, Sonia Tayeb Cherif

---

2. Problem Definition

This assignment implements the Intelligent Scissors algorithm for interactive image segmentation. The technique enables users to trace object boundaries by clicking seed points, with the algorithm automatically finding optimal paths along strong edges. Our goal was to create a functional tool that processes grayscale images and outputs binary segmentation masks separating foreground objects from backgrounds.

---

3. Methodology

3.1 Edge Cost Computation

We designed a cost function combining three image features to guide path selection toward strong boundaries:

Gradient Magnitude - Sobel operators compute directional derivatives, with strong edges receiving low costs:

```
grad_x = Sobel(image, dx=1, dy=0)
grad_y = Sobel(image, dx=0, dy=1)
gradient_mag = sqrt(grad_x² + grad_y²)
```

Laplacian Detection - Identifies rapid intensity changes marking edge locations.

Weighted Combination - Final edge cost balances features empirically:

```
cost = 0.43(1 - gradient_mag) + 0.43(1 - laplacian) + 0.14
```

Lower costs indicate stronger edges that the algorithm prefers when finding paths.

3.2 Dijkstra's Shortest Path

Our core algorithm implementation uses 8-connectivity neighborhoods and priority queue optimization (heapq) for efficient minimum-cost node selection. Starting from each seed point, we expand outward computing cumulative path costs until reaching the destination. Parent pointers enable path reconstruction by backtracking from endpoint to start. The algorithm runs in O((V+E)log V) time complexity.

3.3 Interactive Interface

The GUI provides real-time visual feedback during segmentation. Yellow paths preview potential routes as the mouse moves, green paths show confirmed segments, and colored markers distinguish seed points (red for first, blue for others). Users control the workflow through mouse clicks and keyboard shortcuts (C-close, R-reset, S-save). The dark theme interface minimizes eye strain during extended use.

3.4 Binary Mask Generation

After users close the contour, we generate binary masks by: (1) drawing all paths on a blank mask, (2) detecting closed contours with OpenCV's findContours, (3) filling the largest contour to create solid foreground regions, and (4) saving results where 255 represents foreground and 0 represents background pixels.

---

4. Implementation

Language: Python 3.x
Core Libraries: OpenCV (image I/O, gradients), NumPy (arrays), Tkinter (GUI), PIL (display), heapq (priority queue)
Custom Code: Dijkstra's algorithm, edge cost functions, path visualization, event handling (~500 lines total)

The code separates concerns through two main classes: IntelligentScissors handles algorithm logic while IntelligentScissorsGUI manages user interaction. All functions include documentation explaining parameters and behavior.

---

5. Results and Discussion

Application Interface:

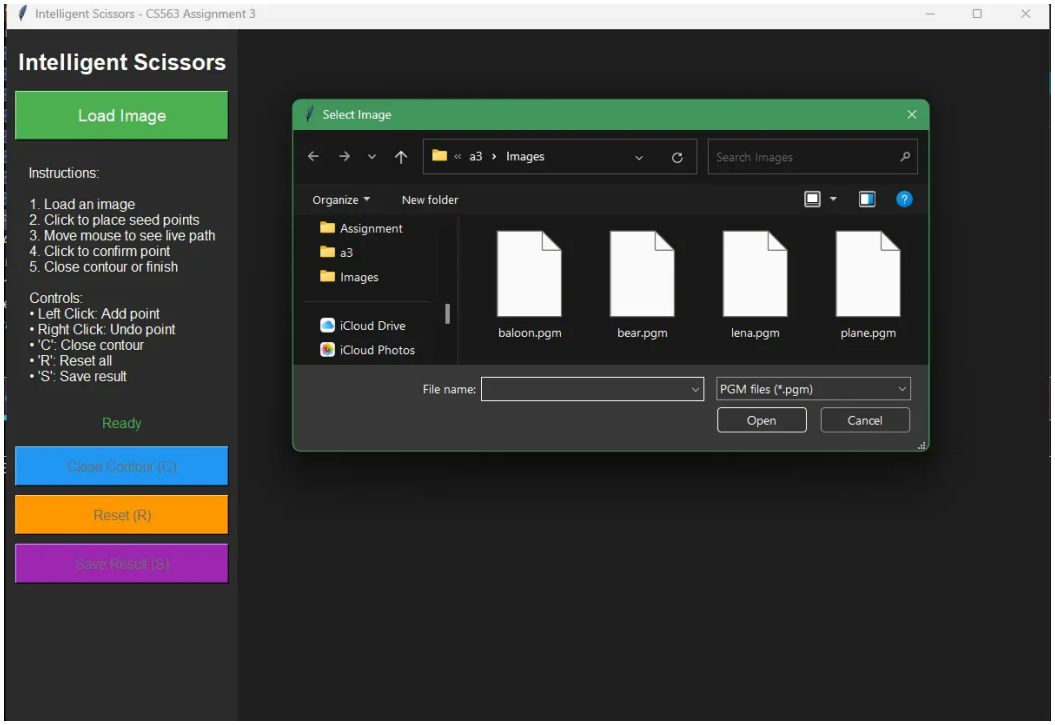

Figure 1 - Main GUI at startup showing welcome dialog and control panel

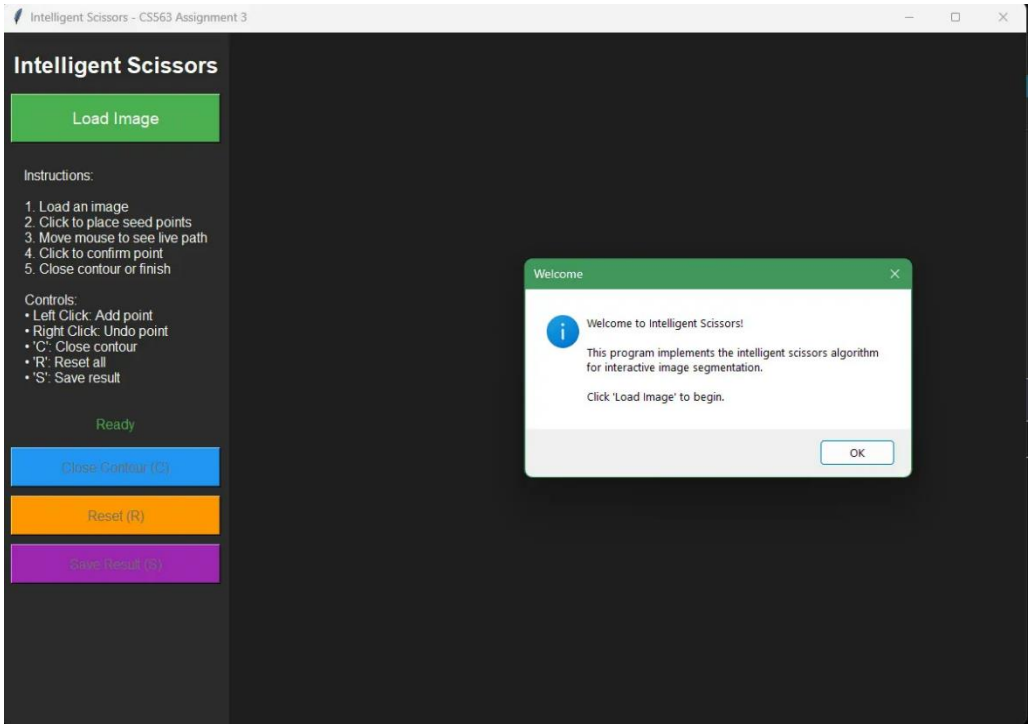

Figure 2 - File selection dialog showing Images folder with test files

Segmentation Results

We tested the implementation on four standard images, demonstrating effectiveness across varying edge complexity and contrast levels.

Bear Image:

Figure 3 - Bear original

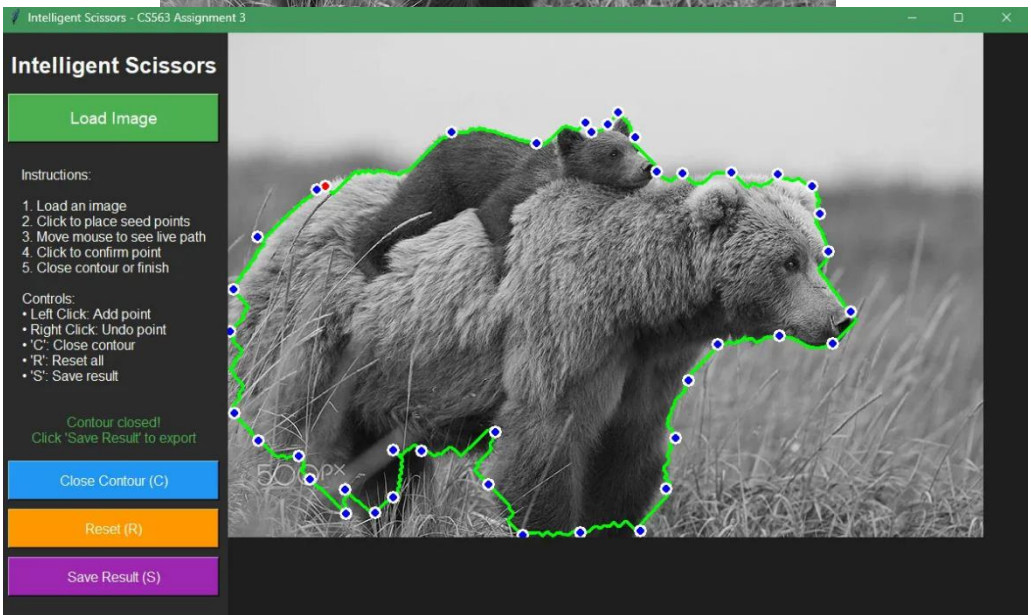

Figure 4 - Bear segmented with green path

Figure 5 - Bear binary mask

The bear demonstrates handling complex fur boundaries and textured backgrounds. Despite challenging edge conditions, the algorithm traced the outline accurately using approximately ~40 seed points (I COULD DO LESS BUT THIS IS MOST ACCURATE). The live-wire feedback helped users navigate areas where foreground and background textures created ambiguity.

Balloon Image:

Figure 6 - Balloon original

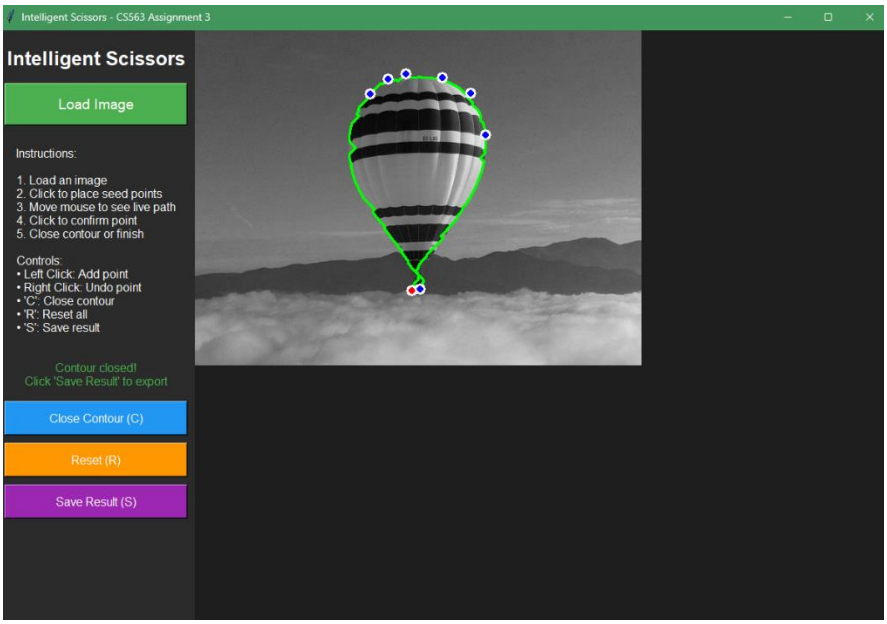

Figure 7 - Balloon segmented

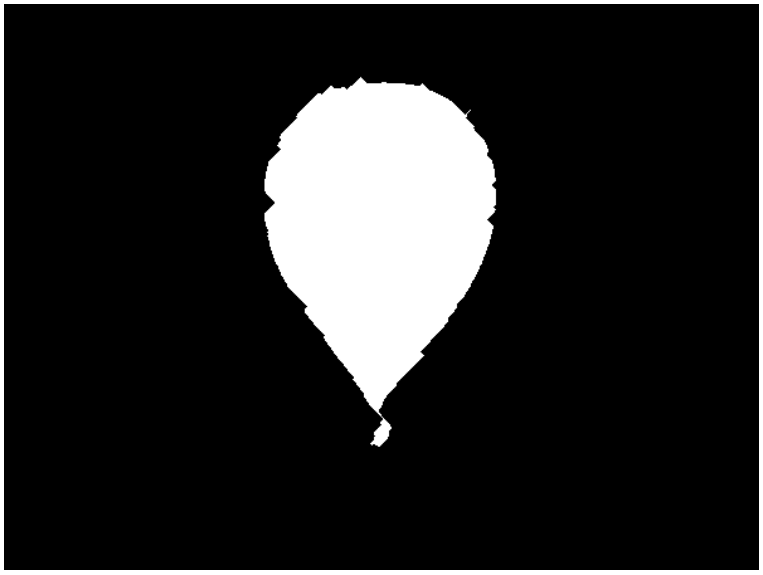

Figure 8 - Balloon binary

High contrast between balloon and background made this an ideal test case. Clean edges required only ~8 seed points for complete segmentation. The thin string attachment was captured successfully, validating algorithm performance on narrow structures.

Plane Image:

Figure 9 - Plane original

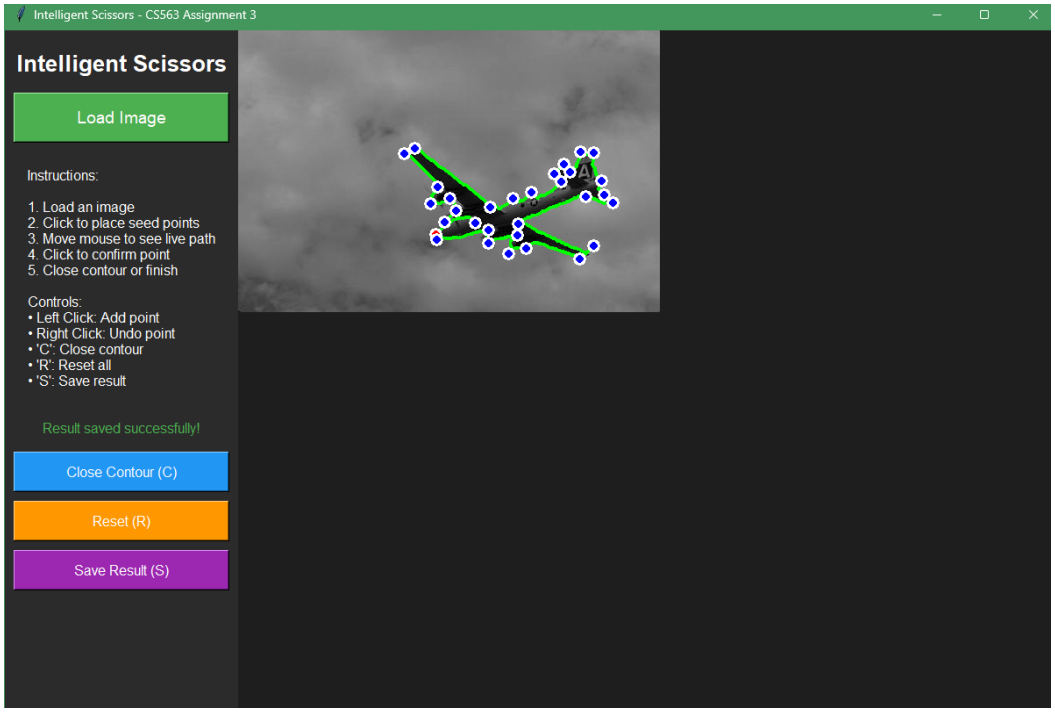

Figure 10 - Plane segmented

Figure 11 - Plane binary

Thin wing and tail structures challenged the algorithm but were successfully traced. ABOUT ~30 seeds needed for most accurate. Angled edges required careful seed placement, though real-time preview made accurate tracing straightforward even for complex geometries.

Lena Image:

Figure 12 - Lena original

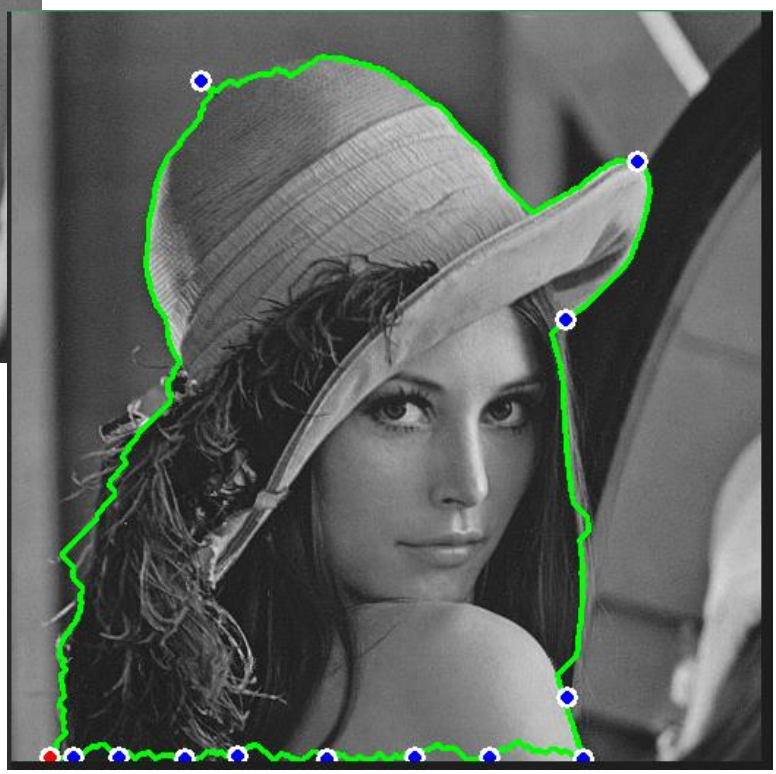

Figure 13 - Lena segmented

Figure 14 - Lena binary

The Lena image required only 13 seed points for successful segmentation, primarily concentrated along the bottom edge to achieve a precise horizontal cut separating the torso. The hat and shoulder boundaries needed fewer points due to strong edge contrast with the background. The algorithm effectively handled challenging regions like the feathered hat decoration and hair texture, producing a clean binary mask that accurately captured the subject's silhouette despite moderate contrast variations.

Performance Analysis

Edge cost precomputation takes 0.1-0.3 seconds per image as a one-time initialization. Real-time path finding during mouse movement completes in under 0.05 seconds, providing smooth interaction. Memory usage scales linearly with image dimensions, handling test images up to 1024×1024 efficiently.

The algorithm follows strong edges accurately with minimal deviation. High-contrast boundaries require fewer seed points while low-contrast regions need denser placement. The live-wire preview significantly improves user accuracy by showing path behavior before committing points. Undo functionality enables quick correction of placement errors.

Challenges Encountered

PGM Format Compatibility: OpenCV occasionally failed loading PGM files. We implemented fallback loading through PIL, converting to NumPy arrays for processing compatibility.

Cost Function Tuning: Initial uniform weighting produced suboptimal paths. Testing various combinations on sample images led to current 0.43/0.43/0.14 ratios that balance gradient and Laplacian contributions effectively.

Smooth Boundaries: 4-connectivity produced jagged paths. Switching to 8-connectivity with diagonal neighbors generated natural curved contours matching object shapes.

Mask Filling: Direct pixel-by-pixel filling proved unreliable. OpenCV's contour detection and drawing functions provided robust filling for closed regions.

---

6. Conclusion

This project delivered a complete Intelligent Scissors implementation meeting all assignment requirements. The system combines efficient algorithms with intuitive interaction, successfully segmenting all test images. Key

accomplishments include custom Dijkstra implementation, multi-feature edge costs, professional GUI with real-time feedback, and robust binary mask generation.

Future work could explore magnetic lasso functionality with automatic seed suggestion, multi-object segmentation support, machine learning edge cost refinement, and export to vector formats. The project demonstrates that classical computer vision algorithms remain highly effective for interactive tasks when paired with thoughtful interface design.

References

- Course lecture materials including notes, textbooks and “Intelligent scisors - Paper to read” File
- Mortensen & Barrett (1995). Intelligent scissors for image composition. SIGGRAPH.
- OpenCV Documentation: https://docs.opencv.org/

Appendix: Usage Guide

(if needed, additional full user guide information has been given in readme.md file in the project folder)

Requirements: Python 3.7+, opencv-python, numpy, Pillow, tkinter

Installation: `pip install opencv-python numpy Pillow`

Running: `python main.py`

Workflow: Load image → Click seed points → Watch live preview → Close contour (C) → Save result (S)

Tips: Start with high-contrast images, place points at corners/curves, use right-click to undo, ensure complete closure before saving.