CS203B Project-Intelligent Scissors

Project member: Yitao Zeng, Haoyang Pan, Ziheng Zhang

github：https://github.com/24zen0/Intelligent_Scissors_for_dsaa

Readme

Load: Press "L"(in English keyboard)

Switch-over with cursor-snap: Press "P"(in English keyboard)

Inputting grids: Press "G"(in English keyboard)

Switch-over with path cooling: Press "C"(in English keyboard)

Image Process

Grayscale Conversion

Algorithm

Using BT.601 standard to convert the three-dimensional RGB components into one-dimensional grayscale for image processing: $\text{Gray} = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$

Implement

- Input: Color pixel matrix `int[][] pixels` (ARGB format)
- Output: Grayscale matrix `int[][] grayPixels`

Sobel Operator

Algorithm

Detect image edges using Sobel kernels for horizontal (Sx) and vertical (Sy) gradients:

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

- Horizontal gradient: $G_x = \sum_{i=-1}^{1} \sum_{j=-1}^{1} SobelX[i+1][j+1] \times P(y+i, x+j)$

- Vertical gradient: $G_y = \sum_{i=-1}^{1} \sum_{j=-1}^{1} SobelY[i+1][j+1] \times P(y+i, x+j)$
- Gradient magnitude: $G = \sqrt{G_x^2 + G_y^2}$

Implement

- Input: Grayscale matrix `int[][] grayPixels`
- Output: Gradient magnitude matrix `double[][] sobelMatrix`
- Optimization - Border Pixel Handling:

To ensure that 3×3 neighborhood lies entirely within the image, avoiding out-of-bounds access.

Outer loop `y` ranges from 1 to `height - 2`; Inner loop `x` ranges from 1 to `width - 2`

Fg Normalization

$$f_G = \frac{G_{\max} - G}{G_{\max}}$$

This formula normalizes the gradient magnitude G to the interval $[0, 1]$, serving as an indicator for measuring pixel characteristics.

Cost Matrix Construction

Cost Function

$$C(x, y) = \frac{1}{1 + G(x, y)}$$

The function calculates edge costs based on gradient magnitude G, encouraging the algorithm to follow low-cost (high-gradient) paths, where higher gradients yield lower costs, guiding the path along object boundaries.

Implement

- Input: Gradient magnitude matrix `double[][] sobelMatrix`
- Output: Global static cache `costMatrix`
- The `costMatrix` is generated in one go within `processImageToSobel()`, enabling subsequent calls to `costFunction()` for direct access.

Gaussian Blur Preprocessing

Purpose

After directly using the Sobel algorithm to compute the gradient of the image and assign it to G, it was found that the edge detection accuracy was poor. Thus, Gaussian blur processing was incorporated to reduce image noise and smooth the image, providing a better foundation for subsequent edge detection.

Implement

$G(x, y) = (1/(2\pi\sigma^2)) * e^{(x^2+y^2)/(2\sigma^2)}$

The 2D Gaussian function implements uses 3×3 kernel with σ=0.8, balancing efficiency and blur quality.

Algorithm

Main Loop

Structure

The code employs a standard game loop architecture, which is divided into three main phases:

代码块

```
public void run() {
    init();
    loop();
    cleanup();
}
```

Core Update Logic

Pseudocode

代码块

```
PSEUDO-CODE:
FUNCTION logicUpdate()
    IF hasSeedPoints THENstart = lastSeedPointend = snap(currentMousePos)
        path = AStar(start, end, costMatrix)
        CONVERT path nodes to points
        UPDATE line segments
        ADD to preview buffer
    END IF
END FUNCTION
```

Implement

- Mouse alignment via `snap()`
- A* search on gradient cost matrix
- Path conversion to preview buffer

A* Pathfinding Algorithm

A* is a heuristic search algorithm widely used in pathfinding and graph traversal. It combines the advantages of Dijkstra's algorithm (guaranteeing shortest paths) and Greedy Best-First Search (improving efficiency through heuristics).

Key Formula

f(n) = g(n) + h(n)

- g(n): Actual cost from start node to current node
- h(n): Heuristic estimated cost from current node to goal

Heuristic Function

We use Euclidean distance in this implementation:

```
代码块
1   Math.sqrt(Math.pow(x1-x2, 2) + Math.pow(y1-y2, 2))
```

Implement

- The first method we use is `findpath` :here we input the mouse-clicks and cursor current positions as start and end coordinates, and the method will return `reconstructPath(current);` ,which is an Arraylist of Link<Node>.
- The second method is `heristic` : where the method calculates the estimated cost to the goal.
- The third method is `reconstructPath` : where the method returns the points from the `Node` to get the final found path.
- The fourth method is `convertNodesToPoints` :where the method returns the form of `Link<Node>` into `Link<Point>` , only in this way can the point sequence be stored in `pointlist` for further line drawing.

Drawing

Drawing lines:

- About the design of preview lines and saved lines: `addNewPointsSave()` and `addNewPointsPreview` : these two respectively represent the arrays that will further on be drawn. There is a slight difference between these two: The preview is a sequence of points making it `Link<Point>` , while the Saved is a sequence of lines,making it `Link<Link<Point>>` .
- `renewLine` :collects the points that A* returns and concatenates them into a line.
- `drawPointsAndLinesPreview()` :this draws the circle which represents the seedpoints and also the lines that are preview (which means the lines will rapidly vanish and reappear).
- `drawLinesSave()` : this draws the saved lines(either by clicking or path cooling), different from `drawPointsAndLinesPreview()` this method will be permanently kept on the canvas.

Closed loop:

After the positions of two clicks by the user are close enough, it determines that the path is closed and then cuts out the image within the path for output.

Cursor snap

Principle

This algorithm dynamically locates optimal alignment points by traversing the pixel neighborhood within user-specified radius and computing gradient feature costs in real-time.The cursor snap algorithm can determine the optimal position where the cursor should snap, thus achieving precise alignment with edge features.

Implement

- The method call is implemented in the loop using the following code. After multiple rounds of debugging, it was found that a radius value of 13 yields relatively good results.

```
代码块
1   Point end = snap((int) getPosition().x, (int) getPosition().y,13);
```

- In the mouseClickCallback method, the position where the mouse cursor clicks is converted into a snapPoint, and this point is recorded in the seedPoints list.

```
代码块
1   Point rawPoint = snap((int) getPosition().x, (int) getPosition().y,13);
```

Results

It can be noticed that the cursor is only near the edge, but the snapPoint can snap to the actual edge.

Path cooling

General view:

The major aim of path cooling is to lower the amount of time you click the mouse, and the computer will automatically help you save the stable points and lessen the complex and tedious operation.

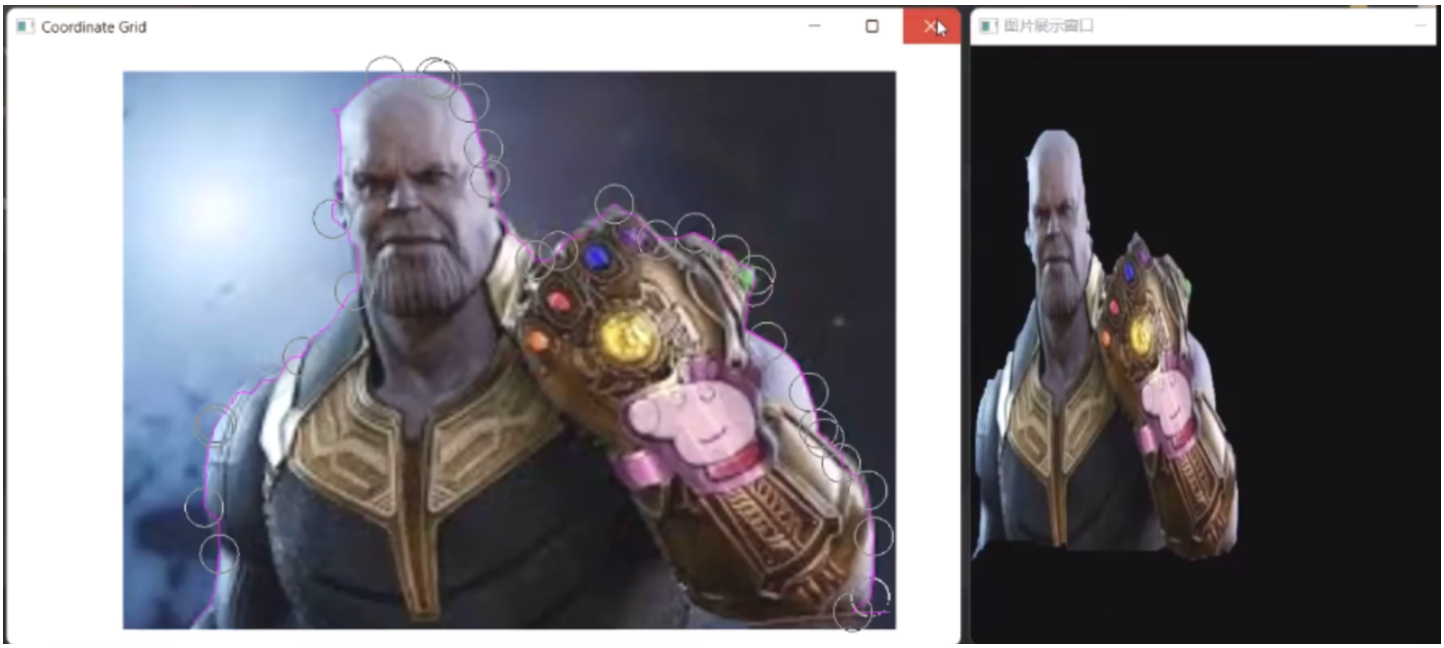
Function and usage:

Automatic recognition of stable points: Could recognize the stable points from the preview lines and automatically save them into the `pointListListSave`.

Automatically saving and drawing the stable points onto the canvas: after receiving the points from `pointListListSave` and detecting the end of stable segments, the method `drawLinesSave` could turn the previewed drawn lines into permanent drawn lines.

The method of detecting stable segments is through recalling the pointListListSave, if a point remains unchanged in 5 iterations, then it is considered stable and will be saved.

Results:



The circles are the saved part. It could be viewed in the video that the system has automatic saving functions.

GUI Design

General view

We keep a simple GUI design with neccessary function and guidance.

The initiation frame desplays a guidance with brief constructions below.

Function and usage

1. Click "L" : load your image (load your image at scr/*), in English input method.
2. Click "1" : load sample image.
3. Ctrl + mouse wheel zoom : scale the image size.
4. Mouse wheel zoom : scale the mark size of seedpoints you choose or auto-generated.
5. Drag the window's edge： stable to window size change.
6. Useful model :
    1. handMode： click "H", then you can move the picture; click again to quit.
    2. GridMode : click "G", then a grid frame show up to help visulizing the pixels.
7. Specific manual:

    1. Place/remove seedpoint： click LEFT mouse button, it will marked the nearest pixel as a seedpoint; click RIGHT mouse button, it will remove the last marked seedpoint if your mouse is approxomately on it.

2. When you click "Enter" or click the first seedpoint (start), the path will close automatically.==un==

3. Click "Ctrl + Z" at anytime if you want to withdraw a movement.==un==

4. Push "Enter" again, then the segmented picture will be shown in another window.==un==