In this project, I reinvented the snake algorithm, which is used to capture the outline of an object given an approximate outline defined by a user.

1 How to Use

Place the main.py file in the terminal's current directory. Type in "python3 main.py".The program will prompt for an image file.

After selecting the image, the program will prompt for an approximate contour.

After drawing all the points, click "finish". The program will do some preprocessing. Then the program will prompt for demonstration of the algorithm. Click "Ok".

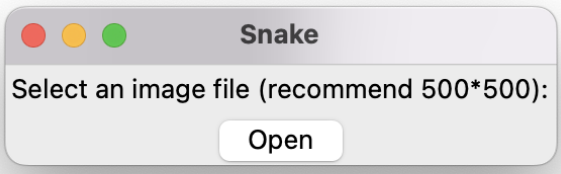

Figure 1: Select an image

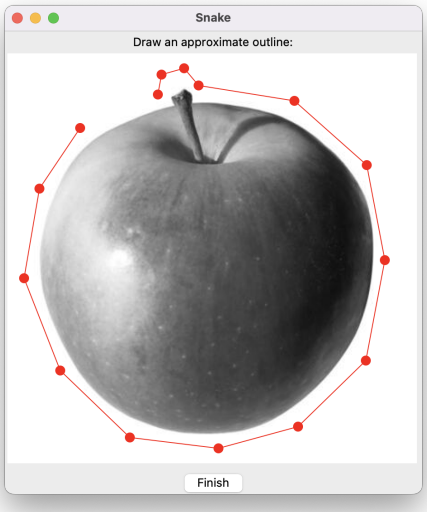

Figure 2: Draw an approximate contour

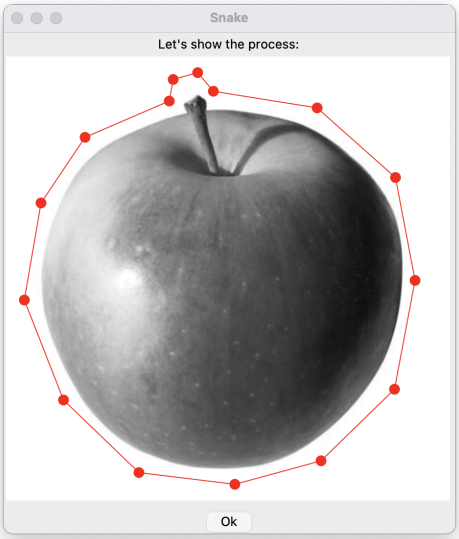

Figure 3: Click ok

2 Results

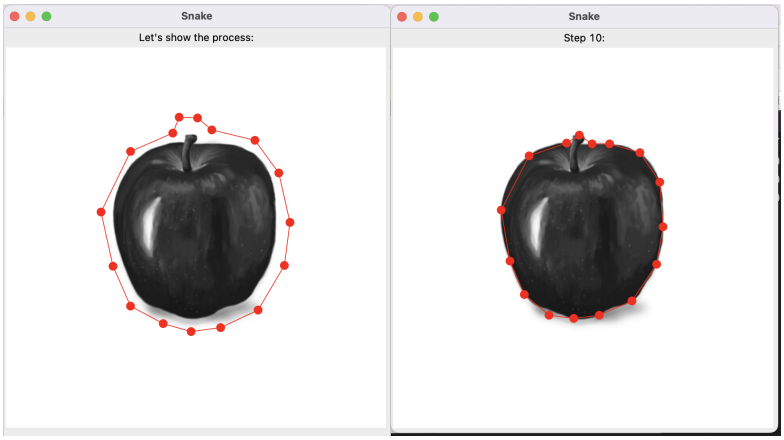

Figure 4: result 1

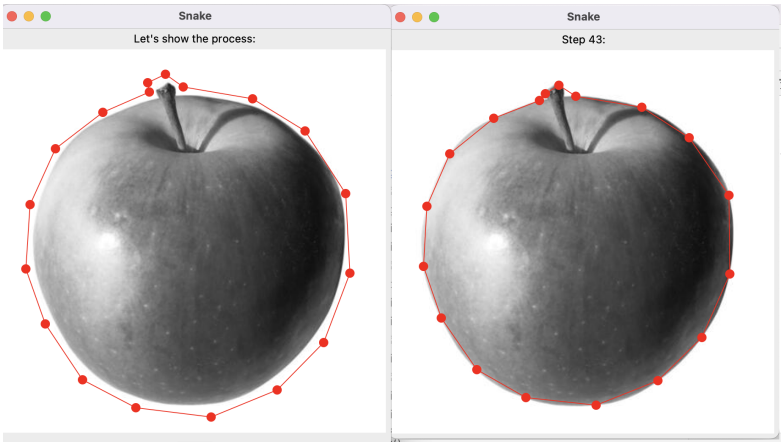

Figure 5: result 2

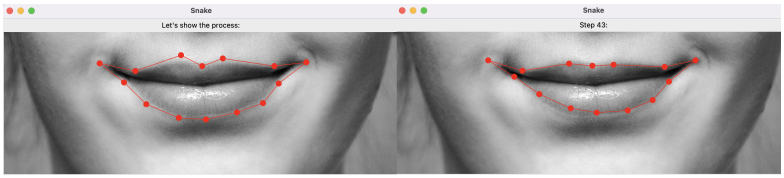

Figure 6: result 3

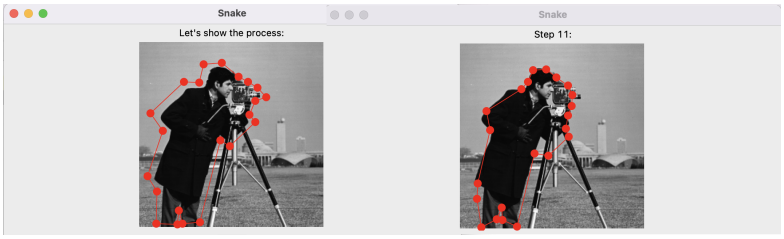

Figure 7: result 4

3 Major Changes

I made three major changes to the original snake algorithm. Two are about the energy function. One is about the window size.

3.1 E_{cont}

The original formula used is $E_{cont} = (\bar{d} - ||p_i - p_{i-1}||)^2$ for two consecutive points p_i and p_{i-1}. One problem with this is that it will completely ignore the original outline specified by the user, which contains important information. Another problem is that it will try to make the points on the contour even spaced, ignoring the fact that some finer details may require more points to capture.

So I propose an alternative that will try to make the contour more proportional to the original contour specified by the user. Before the first iteration, I record the initial length o_{i-1} between two consecutive points p_i and p_{i-1}. The average proportion, $\bar{prop}$ is initially set to 1. For each iteration, the energy is calculated as $E_{cont} = |1 - \frac{||p_i - p_{i-1}||}{o_{i-1} \times \bar{prop}}|$. After each iteration, the average proportion is calculated as the sum of the proportion(current length between two points divided by the original length) divided by the number of points.

The effect of this is that the snake will be forced to shrink more in proportion.

3.2 E_{curv}

The original formula used is $E_{curv} = ||p_{i-1} - 2p_i + p_{i+1}||^2$ for three consecutive points p_{i-1}, p_i, and p_{i+1}. The problem with this is that it only forces the middle point of the three consecutive points to stay in the middle (in other words, it will encourage three points in a line). As a result, 90-degree corners are likely to form. In the original algorithm, the authors try to find these corners between iterations and set the β for those corners to zero to levitate the problem.

So I propose a softer alternative to that using the angle between lines. The new formula is $E_{curv} = 1 - \frac{(p_i - p_{i-1})\dot{(}p_{i+1} - p_i)}{||p_i - p_{i-1}||||p_{i+1} - p_i||}$. This formula will not force the middle of the three points to stay in the middle but also encourage an obtuse angle between lines. Also, this will prevent 90-degree corners from forming.

3.3 Window Size

In the original algorithm, the window size is a fixed number (3, 5, 7, etc).

To make the points stay where the gradient magnitude is larger, I propose adjusting the window size of a point based on the gradient magnitude of that point. If the gradient magnitude is large, the window size should be smaller; otherwise, the window size should be larger. For a 500x500 image, the formula used for window size is $w = w_{base}(-1.5m^2 + 2)$ with w being the window size, w_{base} as the base (=2), and m is the gradient magnitude at that point.

4 Parameter Choice

The percentage of points not moved to end the iterations is 0.85. α for E_{cont} is set to 1, β for E_{curv} is set to 1, and γ for E_{image} is set to 1.2.