Greedy Snakes

Renaud Bourassa
20321718

December 5, 2011

1 Introduction

Fitting curves to objects is a problem that has many applications in various fields ranging from medical imaging to security. One of its main use is in video tracking, where the position of an object must be determined from frame to frame. Snakes, or active contour models, try to solve this problem by generating splines that move within the images until they stick to an object boundaries. First proposed by Kass, Witkin and Terzopoulos [3], many algorithms have since been proposed to improve on their original idea [1][5][6]. This report implements and analyzes the algorithm put forward by Williams and Sha [5]. In their paper, they describe a new algorithm that uses a greedy approach to energy minimization. This algorithm is not only faster than previous approaches, but it also introduces improvements to the energy functional itself by reformulating some of its terms, improving its accuracy.

2 Overview

2.1 Energy Functional

The active contour model, as originally described by Kass, Witkin and Terzopoulos [3], is an energy minimizing model where the energy of a spline under the influence of image forces and external constraints is minimized. Given a vector of points $p(t) = (x(t), y(t))$, $t \in [0, 1]$, that represents the spline, the energy functional for the contour is defined as follow

$$E^*_{snake} = \int_0^1 E_{snake}(v(s))ds$$
$$E^*_{snake} = \int_0^1 (E_{int}(v(s)) + E_{image}(v(s)) + E_{con}(v(s)))ds$$

Where E_{int} represents the internal energy of the spline resulting from its deformation, E_{image} reflects energy gathered from image forces and E_{con} is the energy term corresponding to external constraints imposed on the snake. Since external constraints are user defined and can vary vastly from one application to another, the E_{con} term will largely be ignored throughout the remaining of this report.

2.2 Internal Energy

The internal energy of the spline, E_{int}, is divided in two terms E_{cont} and E_{curv}. The first one accounts for the spline continuity and should grow larger as the spline is stretched. On the other hand, the second one accounts for the spline curvature and should grow larger as the spline is bent. The original equations proposed to compute these terms are as follow [3]

$$E_{cont} = |v_s(s)|^2$$
$$E_{curv} = |v_{ss}(s)|^2$$

As suggested by Williams and Shah [5], E_{cont} can be approximated as follow

$$E_{cont} = (\bar{d} - |p_i - p_{i-1}|)^2$$

This approximation of E_{cont} initially forces the snake to constrict the object, but it also makes sure the points won't cluster on a certain edge and that they are equally spaced. The E_{curv} term can be approximated as follow

$$E_{curv} = |\frac{\vec{u}_i}{|\vec{u}_i|} - \frac{\vec{u}_{i+1}}{|\vec{u}_{i+1}|}|^2$$

Which corresponds to the difference between the vectors in and out of the current point and formed with the neighboring points. Since E_{cont} should ensure the points are equally spaced, E_{curv} can be further approximated by omitting normalization. The resulting term is as follow

$$\begin{aligned} E_{curv} &= |\vec{u}_i - \vec{u}_{i+1}|^2 \\ &= |(p_{i-1} - p_i) - (p_i - p_{i+1})|^2 \\ &= |p_{i-1} - 2p_i + p_{i+1}|^2 \end{aligned}$$

This approximation is computationally efficient and returns larger values for larger differences in vector direction which is a sign of strong curvature.

2.3 Image Energy

The last energy term is the image energy E_{image}. Kass initially suggested that this term should be minimal when the snake stops over image edges [3]. One possible way to extract edges is with a Canny filter, as initially proposed by John Canny [2]. This filter combines a Gaussian kernel G with the derivative of the image I as follow

$$E_{image} = -|\nabla(G * I)|^2$$

The negative sign reverses the energy so that sharp edges are mapped to areas of low energy. The filter can be precomputed by taking the derivative of the Gaussian first and being separable, it can be applied efficiently to any image.

2.4 Weights

The energy functional resulting from these terms can then be weighted arbitrarily to satisfy different needs. Given n points describing the spline, the energy functional can be approximated using the following discrete formula

$$E^*_{snake} = \sum_{i=1}^{n} (\alpha(i) E_{int} + \beta(i) E_{image} + \gamma(i) E_{con})$$

Where α, β and γ can either be global or local to a given point. To ensure the terms are comparable in size, each are scaled to an absolute value between 0 and 1, with 1 being assigned to the local maximum. For image energy, the scale is also pushed up by assigning 0 to the local minimum in absolute terms.

3 Algorithm

3.1 Description

The algorithm employs a greedy iterative approach. At each iteration, a locally optimal state is found and is assigned as the starting state for the next iteration. The algorithm first computes the image energy map since it stays the same throughout the computations. It then starts iterating until each iteration causes only a few points to move. Each iteration proceeds in two steps.

Step 1 - Minimization
The algorithm minimizes the energy of each point individually. For a given point, it looks at a window of a given size centered on the starting point. For each of the points within that window, it computes the value of the energy functional at that location. It then looks for the point that minimizes the energy and updates the coordinate of the point within the spline to that location.

Step 2 - Relaxation
The algorithm then loops through all the newly updated points, calculating the normalized curvature at each point. It then looks for local maxima and if the curvature and the absolute image energy at one of those points is higher than a given threshold, the point is assumed to be a corner and is relaxed by setting its weight β to 0. This effectively prevents the corner from being moved due to its high curvature energy.

3.2 Implementation

This algorithm was implemented using MATLAB [4]. A snake function was written that, given an input image as well as input points as two coordinate vectors, returns two coordinate vectors corresponding to the position of the output points. Many parameters of the algorithm can also be changed, such as the value of *alpha*, *beta* or *gamma*, relaxation thresholds, window size, change tolerance and the filter sigma.

The complete source code listing for this program can be found at the end of the report.

4 Analysis

In order to demonstrate the effectiveness of the algorithm under various conditions as well as its performance qualities, experiments were run using different inputs and parameters. To keep these experiments simple, the data set was limited to grayscale pictures. Also, to make them stand out, energy maps were drawn in colours, going from blue to red for low to high energies.

4.1 Convergence

The first experiment was run on a synthetic image containing no corners or straight edges. The shape was drawn using photo editing software and a Gaussian blur followed by Gaussian noise was applied to mimic the noise found in real images. The aim of this experiment is to explore the behaviour of the algorithm when only the minimization step is applied and no relaxation is required. The input image, as well as its energy map, can be seen in Figure 1.

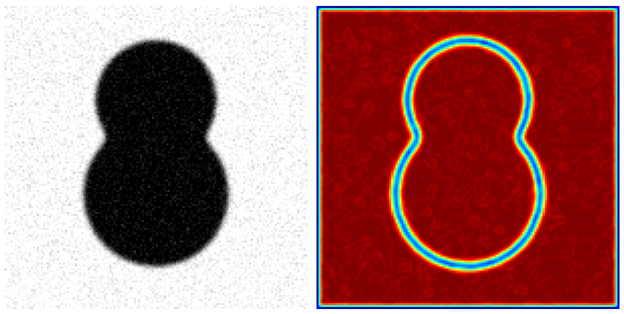

Figure 1: Input image and image energy

As for the parameters of the algorithm, α, β and γ were set to 1.0, 1.0 and 1.2 respectively to give a slight edge to the image energy over the spline internal energy. The window size was set to 7 and the algorithm stops once 2 or fewer points move after an iteration. The results of the experiment can be seen in Figure 2. It takes the algorithm a total of 11 iterations before converging to a minimal solution. At each step, it is possible to see the progression of the algorithm and the energies used to make its decisions.

The first thing to notice when looking at the image sequence, is that a lot more movement is made during the first few steps. By looking at the total energy maps on the fifth row, we can see that during the first few iterations, the energy is much more polarized, with minima occurring near edges or corners of windows. As can be seen in the row above, this is a result of the image energy being insignificant, the data points lying too far away from object boundaries. This causes the snake to be guided mostly by its internal energy. As can be seen in the third row, which corresponds to the curvature energy at each point, the points are pulled inward. The continuity energy maps are also strongly polarized, with each point exerting a pull or a push on its neighbors. The net resulting force thus pulls the snake towards the shape and forces the points to distribute themselves more evenly across the contour.

However, this does not last for long as the points quickly get closer to the shape boundaries. As soon as the fourth iterations, the total energy map can be seen to be much more condensed with minima occurring much closer to the windows center. This is a direct effect of the image energy being the lowest along the edges of the shape, with each window being nearly centered on an edge.

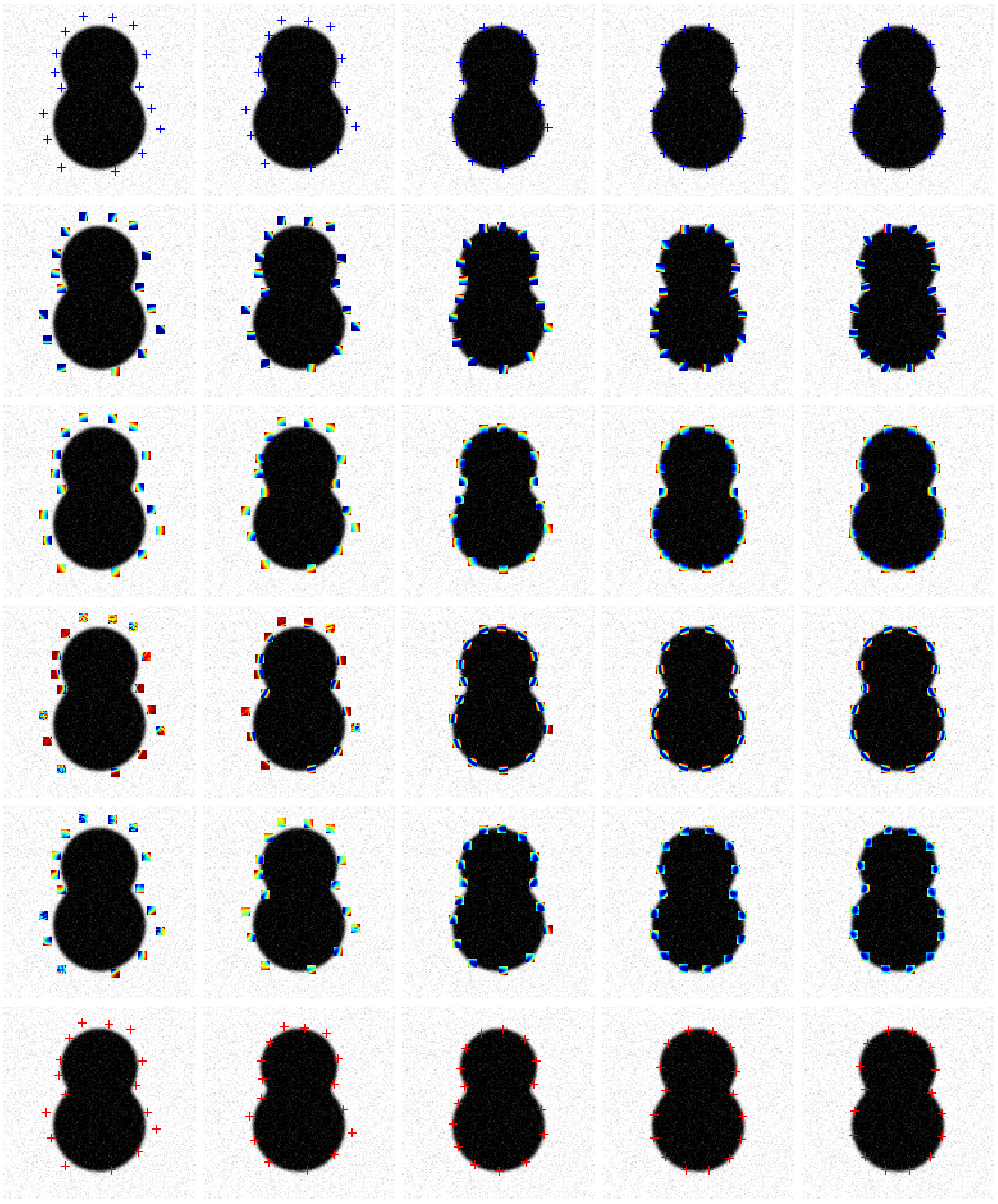

Figure 2: *Input*, E_{cont}, E_{curv}, E_{image}, E_{total} and *Output* for iterations 1, 2, 4, 8 and 11 (final)

The curvature energy still exerts a pull towards the middle, but this pull is easily balanced by the larger γ. The continuity energies are still fairly polarized within each window, but the pull they exert are along the edges. The net result of the last few iterations is thus to balance the continuity term, distributing the points equally along the contour.

The end result can be understood more easily by looking at the final energy maps. From the continuity map, we see that the points now being equidistant, no pull occurs between them. The energy within each window takes the shape of a gradient along the shape edges with the minimum occurring in the middle. On the other hand, the curvature map shows a pull towards the inside for all points except the ones occurring at concavities. Since the shape contains no straight edges, it is never fully minimized due to the pull from both neighbors. However, as can be seen from the image energy map, the points sit comfortably on the shape edges. The result can be seen on the total energy map, with most points sitting on a local minimum.

4.2 Relaxation

The second experiment was run against a synthetic image built using the same technique used for the first experiment. This time however, the shape is made strictly of straight edges to be able to observe the relaxation property of the algorithm. The input image as well as its energy map can be seen in Figure 3.

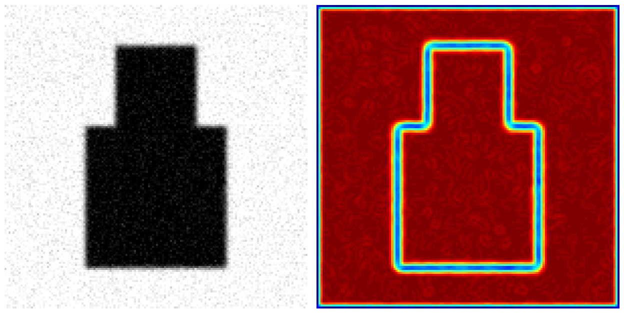

Figure 3: Input image and image energy

The relaxation thresholds were set to 60° for the angle and 0.2 for the image energy. This means that whenever the difference in angle at a point is more than 60° and its absolute image energy is more than 0.2, if it is a local maxima, its β is set to 0. The results can be seen in figure 4. The snake behaves similarly to the first experiment, first moving quickly before slowing down. Near the end, the curvature on straight edges is more balanced then it previously was. However, it is extremely unstable near corners, which is why relaxation is needed.

Relaxed corners are marked with green crosses in the output images. As early as iteration 2, points start being marked as corners and are relaxed. As expected, these occur near actual corner, where the angle between neighboring points is much larger than 60°. The result of the relaxation can be seen in the total energy maps of the last few iterations. Relaxed point do not exhibit the strongly polarized energy features that can be seen in the energy maps associated with the curvature energy and are dominated by image and continuity energies. However, not all relaxed points end on a corner. This is due to the points still being subject to the continuity pull along the edges. The points are thus dragged away from the corners, but this is not an issue. To be relaxed, a point must first be on an edge and once a point is clipped to an edge, the curvature term is not as important.

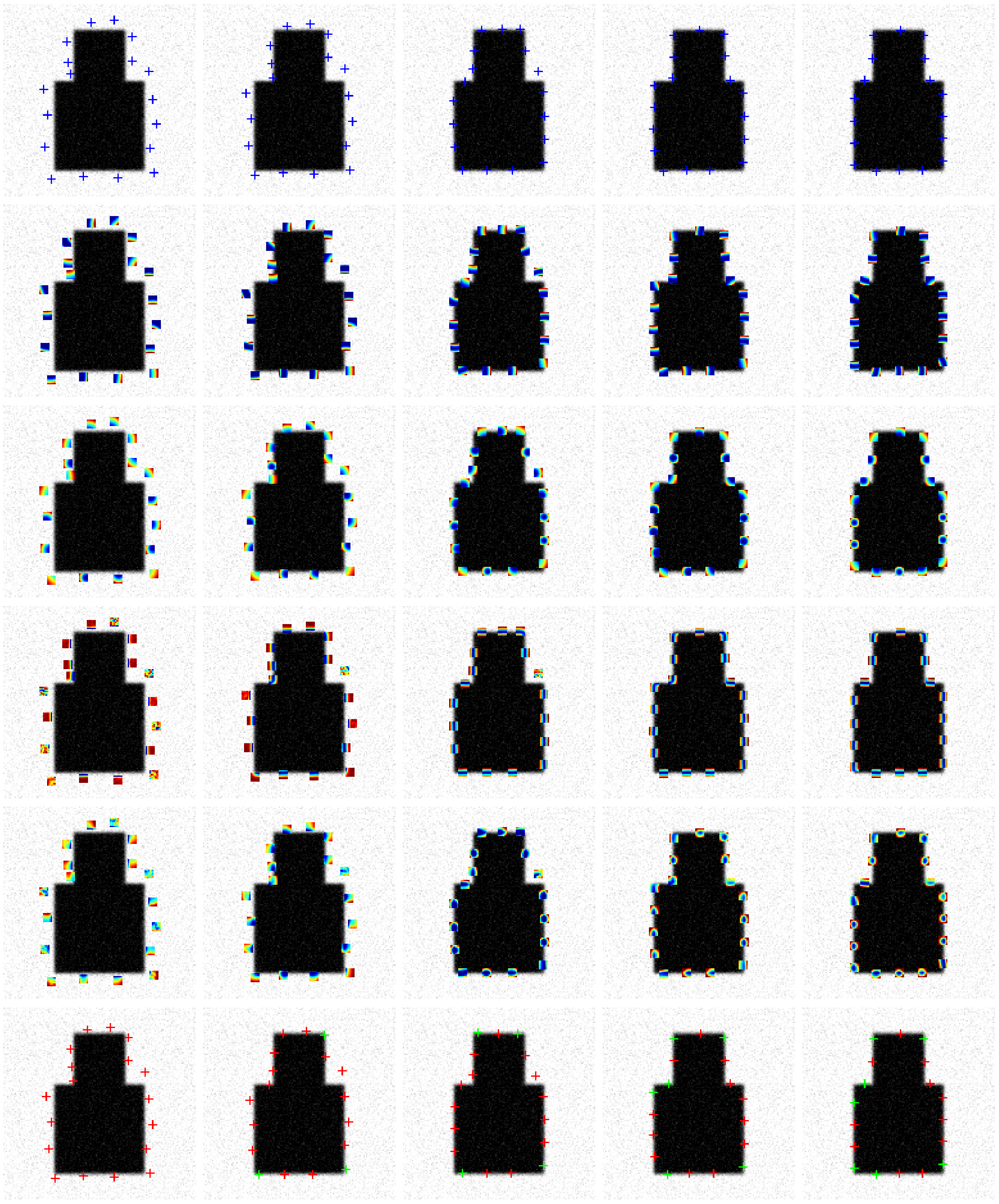

Figure 4: *Input*, E_{cont}, E_{curv}, E_{image}, E_{total} and *Output* for iterations 1, 2, 4, 8 and 15 (final)

It is worth noting that the relaxation step is arbitrary in nature. The threshold values are not universal and can vary from one application to another, depending on what needs to be modeled.

4.3 Weights

This third experiment looks at the effect of the weights on the algorithm output. The same real image was used throughout the experiment as well as the same starting data points. The input to the algorithm as well as the image energy map can be seen in Figure 5.

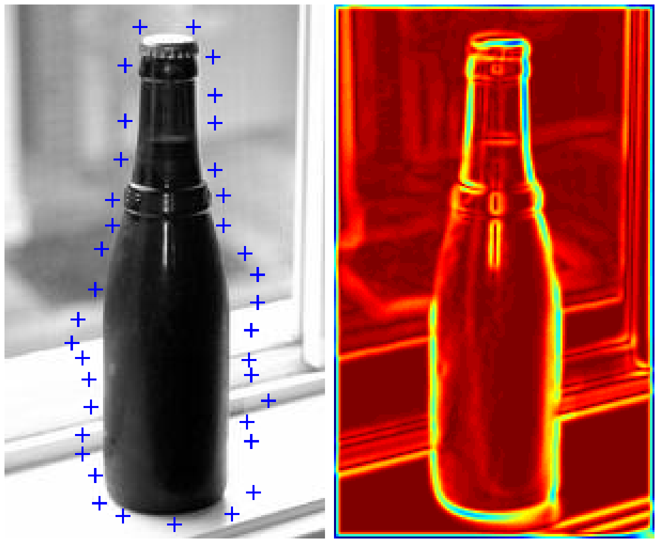

Figure 5: Input image with starting points and image energy

The first sequence of images shown in Figure 6 was generated using the same parameters used in previous experiments. It acts as a control group, using default values for all of α, β and γ. As can be seen, it takes a total of 19 iterations for the snake to converge and the final result contains some noise. Points can be seen to cluster on other edges, such as the window edges, instead of the bottle edges. However, this is limited to a few outliers and most of the snake converges on the bottle.

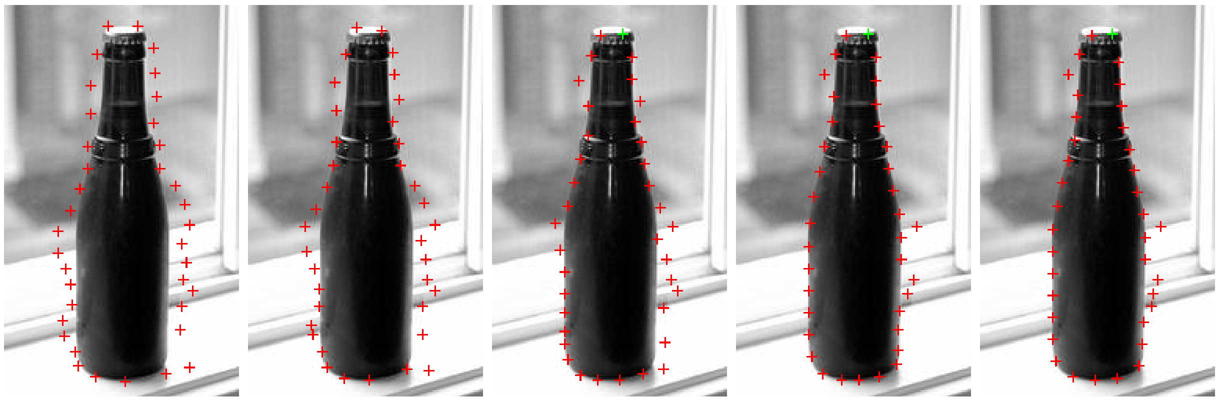

Figure 6: Iterations 1, 2, 4, 8 and 19 ($\alpha = 1.0$, $\beta = 1.0$, $\gamma = 1.2$)

Figure 7 shows the images generated by the algorithm when ran with an α of 3.0 while keeping all other parameters constant. Such an alpha emphasizes continuity over all other terms. The first few iterations are fairly similar to the previous run and it isn't before iteration 4 that we start to see noticeable differences. The direct effect of the higher importance of continuity is to create bubbles where there is a surplus of points. This can be seen most prominently on the right side of the bottle. To maintain a constant distance between the points, some points are pushed away from the bottle. Like a membrane that isn't stretched enough, it breaks loose from the bottle.

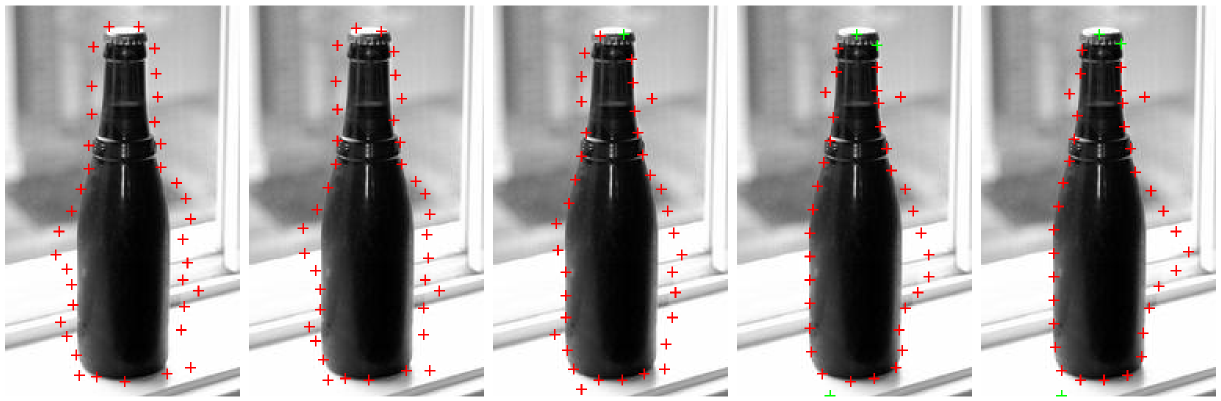

Figure 7: Iterations 1, 2, 4, 8 and 14 ($\alpha = 3.0$, $\beta = 1.0$, $\gamma = 1.2$)

Figure 8 shows the images generated by the algorithm when ran with a β of 3.0 while keeping all other parameters constant. In this case, the algorithm takes much longer to converge and the result does not stick to the edges of the bottle. Right from the start, the algorithm behaves differently, with all points moving to form a more regular shape. The result of each iteration is much smoother. This is a result of the curvature being minimal when the points are aligned. This pushes the snake to move past bottle edges, cutting through sharper corners. The relaxed points make it possible for the snake to form two fairly smooth curves. However, without relaxation, we would expect the snake to minimize the angular differences by minimizing the distance between points, resulting in a small ring of points bound to no edges. It is also important to note that relaxation is less likely to happen since the algorithm will tend to avoid any sharp angle. In the case of objects with irregular shapes, the value of beta can have a large impact on the end result.

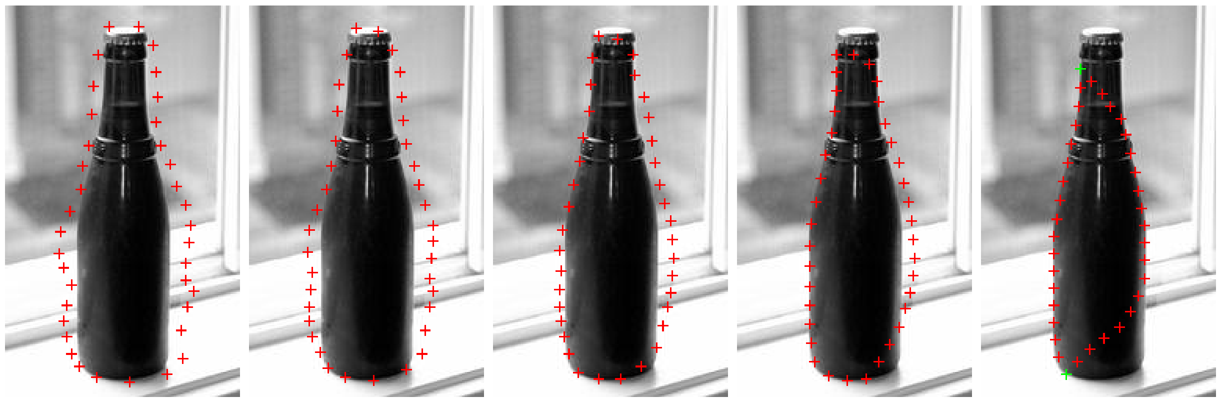

Figure 8: Iterations 1, 2, 4, 8 and 32 ($\alpha = 1.0$, $\beta = 3.0$, $\gamma = 1.2$)

Figure 9 shows the images generated by the algorithm when ran with a γ of 3.0 while keeping all other parameters constant. In this case, the behaviour of the algorithm is much more erratic. The reason is that points will tend to get stuck on areas of low energy resulting from edges owned by other objects in the image. This can be seen all around the bottle, with points stopping on background details such as the border around the window as well as objects outside it. From the image energy map, it can be seen that the window contains a number of areas of low energy that have nothing to do with the bottle. Since the image energy term is made much more important than the spline internal energy, the spline is distorted by points that reach for these minima. It can thus be seen that the more details in the image, the more likely a large gamma is to cause the points to wander away from the target object and the snake to oscillate.

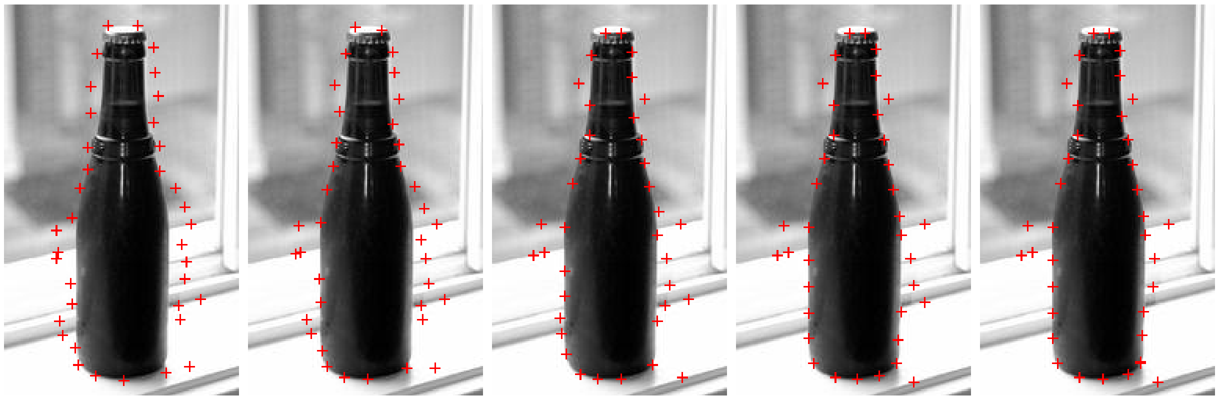

Figure 9: Iterations 1, 2, 4, 8 and 14 ($\alpha = 1.0$, $\beta = 1.0$, $\gamma = 3.0$)

4.4 Performance

Given a fixed number of iterations, the algorithm runtime is linear with respect to the number of points. The reasoning behind this is that each point is inspected once per iteration and a constant number of operations are executed depending on the window size. Assuming the window consists of m pixels, the runtime would thus be $O(nm)$, with n being the number of points in the spline.

However, the number of iterations is rarely fixed. The algorithm should not stop until it reaches convergence and its ability to converge can be affected by such things as the number of points and the window size. To see the effect of both of these factors on the runtime of the algorithm, an experiment was performed using the simple shape shown in Figure 10.

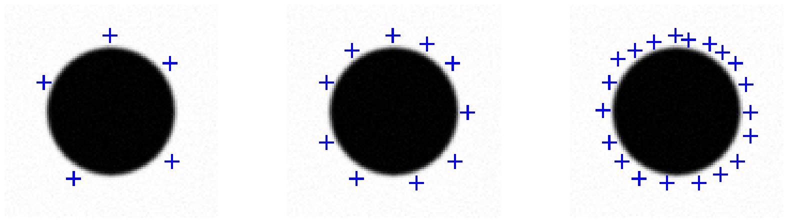

Figure 10: Shape with starting points

Snakes of different sizes using different window sizes were run until no points would move anymore. Table 1 shows the relationships resulting from these factors and the number of iterations while Table 2 shows the differences in runtime. Each run converged to a valid end state.

		Window Size		
		3	5	7
# Points	5	11	8	6
	10	13	7	5
	20	15	11	9

Table 1: Number of iterations before convergence

		Window Size		
		3	5	7
# Points	5	0.121097	0.091076	0.064533
	10	0.233059	0.133159	0.094105
	20	0.510048	0.378405	0.328948

Table 2: Time before convergence

The first thing to note is that fewer points results in faster convergence. Halving the number of points more or less halves the total runtime of the algorithm. This is mainly because the algorithm only has to look at half the number of points at every iteration. The relation with the number of iterations is a little bit less clear. Fewer iterations are observed for fewer points. One reason for this could be due to the last few iterations being limited to balancing points along boundaries. Fewer points means this can be done faster. However, this effect is minimal with only a few points which is why the 5 and 10 version take more or less the same number of iterations.

The relation between window size and runtime is the inverse of what is expected by a simple complexity analysis of the algorithm. This is due to larger window sizes resulting in a smaller number of iterations. With a large window, points are free to move farther away from their starting position. This makes it easier to converge to a stable state, even though more points must be looked at. The result is that the faster convergence in terms of iterations overcome the time taken by the extra computations. However, as the number of points increases, this effect is likely to diminish, since more iterations are required for every window size. This can already be seen with 20 points. The relative difference between a window size of 3 and 7 is already less than with 10 points.

The algorithm runtime is thus highly influenced by the convergence condition. It is also worth noting that there is a trade-off to make between performance and precision, since a lower number of points result in a less precise shape.

4.5 Limitations

This algorithm has a number of limitations that make its use improper for certain applications. The most obvious limitation of the algorithm comes from its disregard for boundary ownership. The internal energy of the snake prevents it from taking absurd shapes to reach far-fetched details, but it does not prevent it from converging on local noise, whether internal or external to the object being modeled. Figure 11 shows an example of this as applied to a picture frame hung on a wall covered with wallpaper.

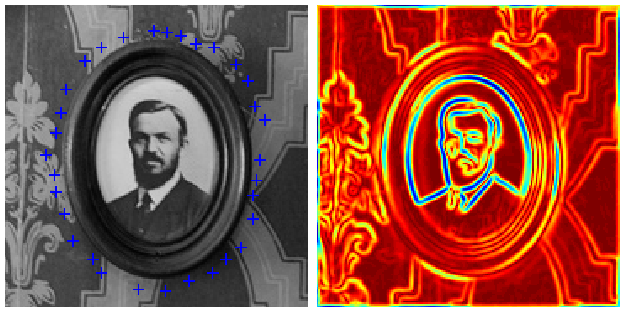
Figure 11: Input image with starting points and image energy

As can be seen on the image energy map, there are many energy minima resulting from the wallpaper around the picture frame. Figure 12 shows the progress of the algorithm on this picture. It shows the snake converging not only on the frame, but also on some of the details surrounding it. This is an inherent flaw in the algorithm. It can be lessen by taking other factors into considerations, such as color, or even a depth map, to augment the image energy with more detailed data.

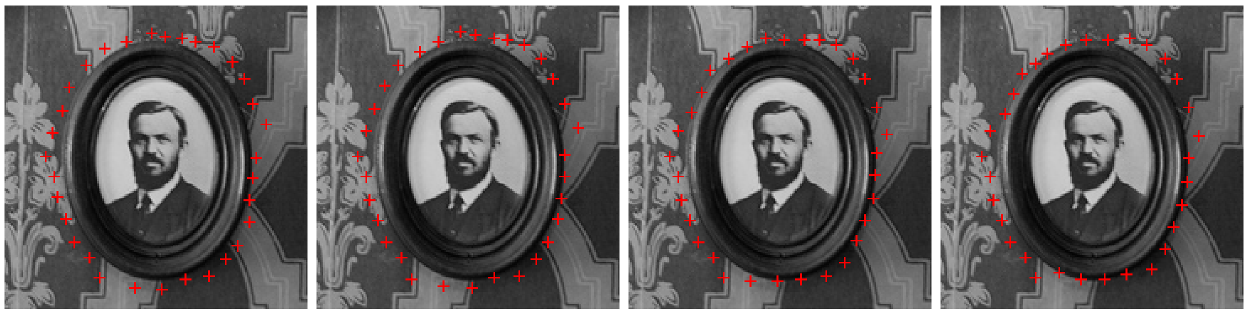
Figure 12: Snake state after iterations 1, 2, 4 and 9

This is also a problem with details within the object itself. Here is an example using a picture of a white dice on a white table. Figure 13 shows the input image and its energy.

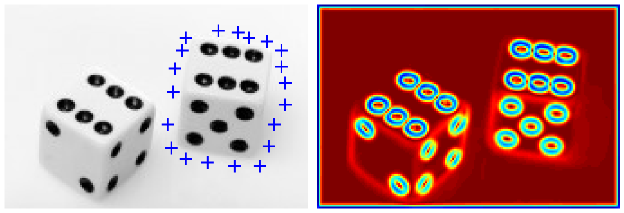
Figure 13: Input image with starting points and image energy

As can be seen from the energy map, the dice edges have a fairly high energy compared to the dots, which are areas of low energy. This result in the snake cutting through corners, as can be seen in Figure 14, to reach for the dots. Even though it starts well by going for the edges first, the pull towards the dots is much stronger due to the low image energy at these points.

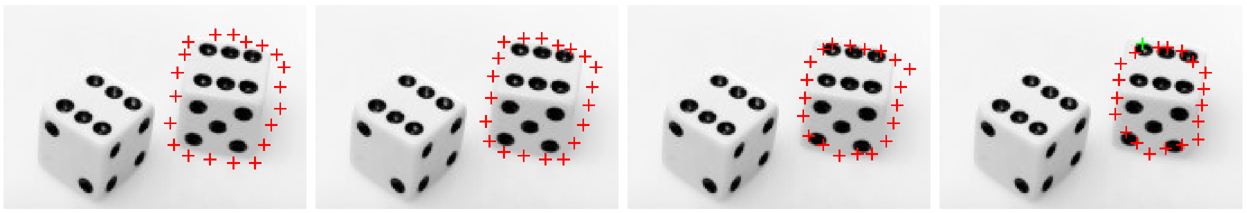

Figure 14: Snake state after iterations 1, 2, 4 and 13

This problem is made even more significant by the normalization of each energy term. If one point encounters even a small amount of noise in the image, it can get stuck on it since it will be normalized to 1 and the combined pull of the continuity and curvature term may not be enough to make it move. This normalization is necessary to be able to compare the different terms, but it also has the undesirable side effect of magnifying insignificant changes.

Another significant limitation of the algorithm worth mentioning resides in trying to fit objects with important concavities. Due to the spline internal energy, points will not tend to go deep within concavities, especially if these are narrow and makes it possible to bridge the gap by keeping the continuity energy fairly low. Figure 15 shows such an image with its energy map.

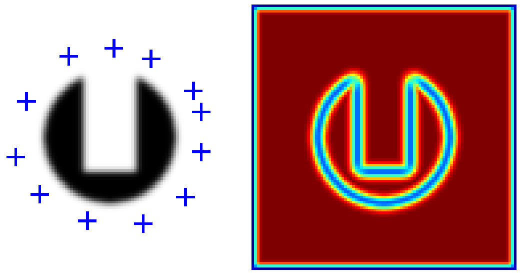

Figure 15: Input image with starting points and image energy

From its starting position, the snake will tend to constrict the object without bending or stretching too much. However, as can be seen in Figure 16, even though corners near the opening are relaxed, it is not enough to get the points to penetrate deeply into the concavity. This is mostly because there is nothing pulling the points farther in. They are already sitting on edges, their distance will always be equal to the gap, and their curvature will only change for the worst by going deeper. The end result is that they sit at a local energy minimum that does not constitute a good fit to the data.

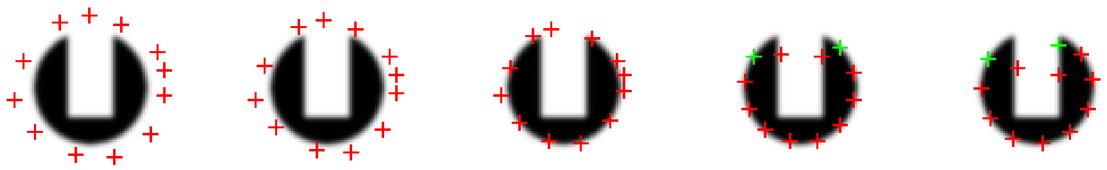

Figure 16: Snake state after iterations 1, 2, 4, 10 and 22

5 Application

One of the many applications of snakes is in object tracking. By first using a computationally expensive search algorithm to find an object in a scene, it is possible to keep track of it using snakes. Since greedy snakes can be computed efficiently, this reduces the burden on the machine. A threshold can be set to decide whether the snake is still valid and only when it wanders away from the object does the search algorithm have to be run again. This is useful in many domains such as security where it can be used to track a person cheaply once discovered by the system.

Assuming the object being tracked does not move by a significant amount between each frame, which is a valid assumption for many types of motions, it is possible to simply reuse the output of the snake on a frame as the input for the algorithm on the next frame. The snake should adapt to the new position of the object by fitting its new contour. This can then be repeated on subsequent frames. One advantage of using this approach is that the snake can also keep track of deformable objects since it does not make any assumption about the shape of the object. It only matches its boundaries.

Such an algorithm was run on a short clip of a deformable bouncing egg being dropped on a table. The initial frame can be seen in Figure 17 with the initial data points. These points were manually selected, but could have been returned by any image search algorithm.

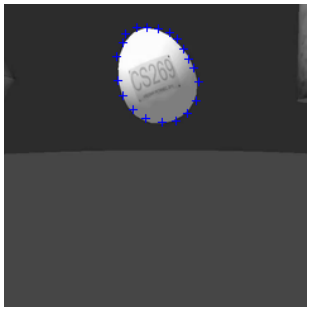

Figure 17: Input image with starting points

Figure 18 shows the progression of the algorithm as it is applied to the video frames. As can be seen on frame 24, even though the egg is deformed, the snake still manages to fit its boundaries. Frame 47 shows the final position of the egg, with the snake accurately fitting its boundaries.

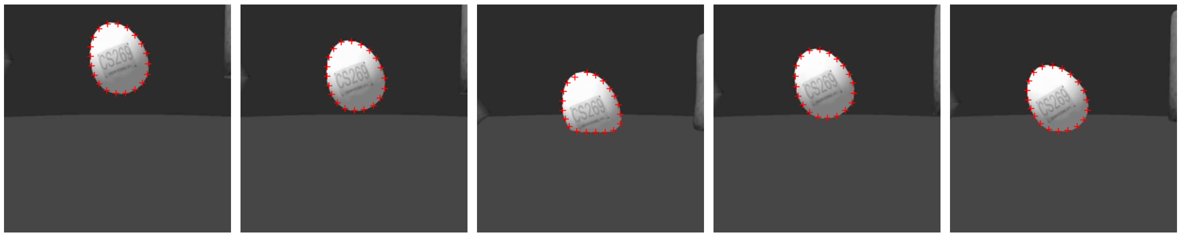

Figure 18: Resulting snakes at frames 1, 12, 24, 36 and 47

The algorithm being implemented in MATLAB [4], it is not fast enough to run in real-time. However, it is already quite fast and it is not hard to see how an optimized C version could achieve real-time performance.

6 Conclusion

A method of computing snakes, or active contours, using a greedy algorithm has been described, implemented and analyzed. Its behaviour as well as its response to changes in parameters has been thoroughly examined. Its performance under a number of circumstances was also analyzed and its limitations demonstrated. Finally, an application of the algorithm to object tracking was demonstrated.

References

[1] A. Amini, T. Weymouth, and R. Jain. Using dynamic programming for solving variational problems in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9):855–867, September 1990.

[2] J. Canny. A computational approach to edge detection. In *RCV87*, pages 184–203, 1987.

[3] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

[4] MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010.

[5] D. Williams and M. Shah. A fast algorithm for active contours and curvature estimation. *CVGIP:IU*, 55(1):14–26, January 1992.

[6] C. Y. Xu and J. L. Prince. Snakes, shapes, and gradient vector flow. *IEEE Trans. Image Processing*, 7(3):359–369, March 1998.