Boundary Snapping for Robust Image Cutouts

Eyal Zadicario[1,3] Shai Avidan[2] Alon Shmueli[1] Daniel Cohen-Or[3]

[1] Insightec Inc., email: {eyalz,alons}@insightec.com
[2] Adobe Systems Inc., email: avidan@adobe.com
[3] Tel Aviv University, email: dcor@tau.ac.il

Abstract

Boundary Snapping is an interactive image cutout algorithm that requires a small number of user supplied control points, or landmarks, to infer the cutout contour. The key idea is to match the appearance of all points along the desired contour to the landmark points, where appearance is given by an intensity profile perpendicular to the boundary. An optimization process attempts to find a contour that maximizes the similarity score of its points with the landmarks. This approach works well in the typical case where the foreground and background differ in appearance, as well as in challenging cases where the subject is clearly perceived, but the regions on both sides of the boundary are similar and cannot be easily discriminated. By enabling the user to define the boundary points directly, the technique is not limited to boundaries that necessarily have to be the most salient or high gradient feature in the region. It can also be used for margin cutout around the boundary. The use of multiple control points along the boundary can handle spatially varying attributes as both foreground and background may change in appearance along the boundary. The final result is accurate, because it allows the user to enforce hard constraints on the boundary directly, at the expense of moderate user labor in positioning the landmark points. Finally, the algorithm is fast, works on a variety of images, and handles situations where the boundary is not obvious.

1. Introduction

Object cutout is the task of extracting, or segmenting, a foreground object from an image. This has become a popular image editing operation with the advent of digital photography and can be found in almost every photo editing software. Yet, despite its intuitive explanation, it proved to be quite a challenging task and a battery of solutions was proposed to solve it. These solutions vary in the way they approach the problem and in the way the user is involved in the process. Image cutout can be interpreted as a region-based or a contour-based process. In a region-based approach, the algorithm aims at learning the statistics of the foreground object and the background so that it is able to find a boundary to distinguish between the two. In contour-based methods, the algorithm aims at finding the contour, which is usually presumed to be the most salient edge in the region. User interaction can also be classified into two categories, that are roughly aligned with the two types of image cutouts discussed above. For region-based methods it is natural to use scribbles, supplied by the user, that indicate which pixels belong to the foreground and which to the background. Contour-based algorithms, on the other hand, typically track the cursor motion, as supplied by the user, and constantly snap the contour to the closest salient edge.

Region-based algorithms do not provide direct control over the boundary location. Instead, the user can update the scribbles in the hopes of getting a better cutout result, but the user cannot specifically mark the boundary. The problem is aggravated when the statistics of the foreground and background are similar. In this case, a considerable amount of user interaction is needed to obtain a good result. Contour-based methods seemingly allow the user a more direct control by tracking the cursor and snapping to the closest salient edge. However, this makes the implicit assumption that the cutout contour is indeed the most salient, or maximum gradient edge in the region. Also, tracking the cursor motion is an online algorithm that is inherently more difficult than batch algorithms such as scribble based methods, that take all the scribbles into account before computing the contour cutout. This often means that user input must closely follow the desired boundary to avoid snapping to erroneous edges.

Given these considerations, we introduce a new variant of contour-based image cutout that relies on a user input taken from the region-based approach. It allows the user to mark the boundary *directly*, just like in contour-based methods, but instead of tracking the entire contour of the object, which may become tedious, it requires just a small

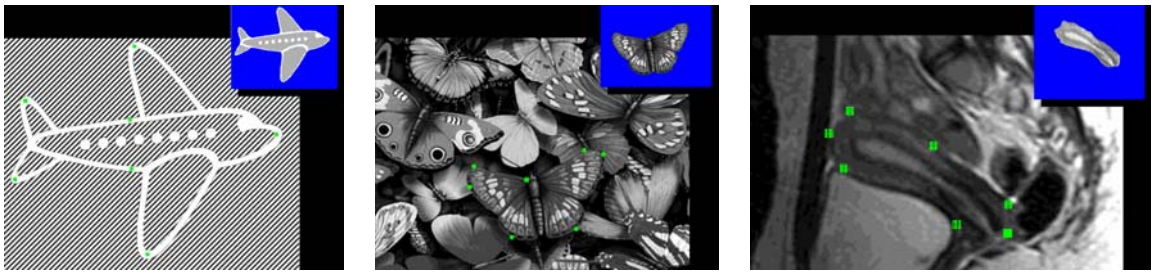


Figure 1. Results of boundary snapping on various image types. The foreground and background exhibit similar texture and yet, our algorithm has no problem finding the correct boundary cutout using a minimal amount of user interaction.

number of control points, like the scribbles used in region-based methods. The system then builds a generative model of the boundary profile and computes, in real time, a boundary whose points resemble in content, as much as possible, the control points. We represent each boundary point as a 1D intensity profile in direction orthogonal to the boundary. The intensity profile is a rich descriptor that improves the segmentation process. We also use it for Margin Cutout, where we segment the object with some margin that is learned automatically from the intensity profile. As a result, the user enjoys complete freedom and control in marking the boundary of the object. Creating a Margin Cutout is useful in a variety of applications (e.g. medical applications) where the cutout must be away from the natural edge defining the boundary. Using a descriptor with higher dimensions is obviously possible [e.g. textures] but we found the simple 1D profile to be sufficient and adequate for the real time interactive performance we set to achieve.

Our technique requires the user to provide a sufficient number of control points to characterize the whole boundary, which in simple cases might be somewhat redundant. However, as can be seen in Figure 1, our results show accurate cutouts of various difficult examples, all of which are achieved in a few seconds of interaction and with very limited user input.

2. Background

Image segmentation has long been known to be a challenging problem and the research on the topic is too vast to be covered here. We focus on the work most relevant to ours.

Our method is focused on interactive boundary-based image cutout. This theme was explored in the past with great success [9, 10, 5, 12]. Typically this is done by constantly following the cursor motion and "snapping" to the nearby edge. Unfortunately, this requires the user to manually track the entire object boundary, albeit roughly, to obtain a satisfactory result. Others have suggested assuming the shape of the object is either known in advance [4] or a previous set of images have been used to train a boundary classifier [8].

Alternatively, one can take a region-based approach [13, 1] and use graph-cuts [2, 14, 3], where the user indicates on a small number of pixels if they belong to the foreground or the background and a discrete optimization is then used to find a boundary based on user input. This approach can show partial results, so the user can review the cutout and then add, remove or modify the input to improve the result. Typically, in region based methods it is difficult to achieve pixel-accurate image cutout, because the user does not precisely define the boundary points directly.

Graph cut methods rely on region labeling to simplify user interaction [2]. The user identifies a sample of object and background pixels. A graph is constructed with weighted edges according to the similarity and vicinity to the identified pixels. The boundary is defined as the min cut through the graph. Since it is based on region labeling, graph-cut methods do not let the user specifically define a desired boundary in areas where the results are locally not satisfactory. This has been recently amended by the Lazy Snapping approach [7]. The approach starts as a graph cut, but once the initial segmentation is obtained it switches to polygon representation that the user can edit. The use of both region and contour based interaction provides a very flexible tool that can be both fast and delicate as needed due to an adaptive combination of region and edge based energy functional. However Lazy Snapping might fail when the foreground and background share a similar texture. In contrast, our approach can handle such cases because it models the boundary directly, albeit, at the expense of increased user interaction.

We take a contour-based approach with a region-based user interface. Instead of requiring the user to track the entire boundary of the object, which can become tedious, we require him to specify only a small number of landmark points, that serve as hints. Instead of indirectly specifying the boundary, by scribbling over foreground and background pixels, we let the user directly mark pixels on the boundary. The result is an extremely fast and easy to use system that gives the user a direct control over the cutout process. In addition to relying on image data in determining the object boundary, we use shape priors, such as smooth-

ness in a manner akin to active contour methods, such as snakes and level sets [6, 11].

It is worth noting the difference between our method and active contour methods. These methods optimize a global objective function that integrates many terms, including region, boundary and gradient orientation, to name a few. Thus offering a less direct control over the process, as well as a lower degree of user interaction. Usually, a single seed point is needed to initiate active contour methods. Also, the relative weight of the various terms is often established empirically. Our method does not assume the boundary to be the salient edge or maximum local gradient in the region. We let the user input define the desired attributes of the boundary and thus are not misled by strong but irrelevant edges. The most important distinction is that we focus on an interactive user interface that allows the user to closely control the algorithm and offer real time feedback. The user does not specify a seed point and let the active contour methods run its course. Instead, the user is constantly in the loop, adjusting the input to achieve the desirable outcome.

3. Boundary Snapping

Our approach starts with some initial user input. The user interactively places few control points where the border of interest is. These landmarks are not necessarily located on local maximal or most significant gradients. The user marks several landmarks that model the characteristics of the boundary and the algorithm then evolves a polygonal snake that is as similar as possible to the landmarks in its behavior. In addition to the image based attributes our algorithm maintains the smoothness of the contour to assure robust results. The result of the cutout is continuously displayed to the user to enable additional input where local results are not satisfactory.

3.1. Evolving polygonal contour

The polygonal snake $P = \{p_i\}_{i=1}^{n}$ consists of an ordered set of n vertices, where $C = \{c_1, \cdots, c_s\} \subseteq P$ denote the control points (i.e. points marked by the user). For each edge defined between the control points c_i and c_{i+1} we add m vertices. Thus, the line connecting each pair of consecutive landmarks is broken into $m + 1$ line segments. The polygonal snake is refined during the evolution of the snake.

The objective function to be optimized, over the image $\mathbf{I}$, is given by:

$$G(P, C, \mathbf{I}) = G_{ext}(P, C, \mathbf{I}) + \lambda G_{int}(P), \qquad (1)$$

where λ is a regularization scalar, and G_{ext} and G_{int} denote the external and internal forces that influence the snake evolution, respectively. The control points C are restricted to remain at the user defined location, while the rest of the vertices in P can move to optimize G. The external component is driven by the landmarks C and the input image $\mathbf{I}$ seeking to place the boundary where it best matches the attributes of the image as defined at the control points. The internal component aims at keeping the boundary smooth by favoring consistent displacement trends in neighboring vertices. The internal forces lead to an iterative process where each vertex is affected by its neighboring vertices. The global term of G evolves during the iterations being a sum of a similar functional for each vertex. A higher contribution of a specific vertex means a higher confidence in the position of this specific vertex on the boundary.

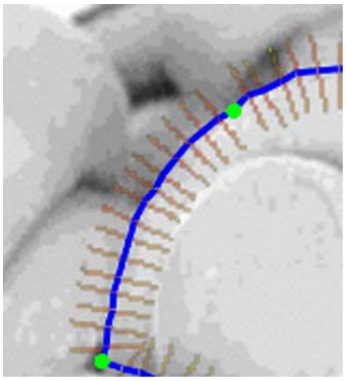


Figure 2. Description of the boundary normals. The algorithm attempts to move the boundary (blue line) such that the intensity profiles of normals along it (light brown line segments) will match those of the landmark points (green points). The intensity profiles of the landmark points are not shown for better visibility.

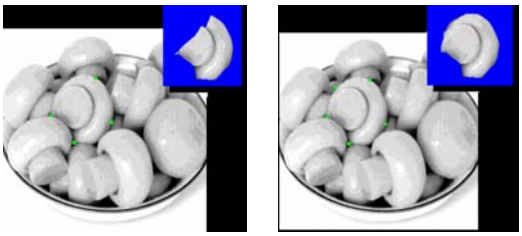
3.2. External forces

The external forces rely on image data to drive the preferred location of boundary. We apply a learning process in which the control points define the attributes of the desired boundary. Given a pair of consecutive control points, $\{c_i, c_{i+1}\}$ we denote the vertices between them by $\{p_k\}_{k=1}^{m} \subset P$. Initially these points are equally placed on the edge $\{c_i, c_{i+1}\}$. Let $\delta(p_k)$ denote a 1D profile along the local normal to the borderline and centered around p_k. The profile is a 1D vector having the underlying pixel values of the image on both sides of the borderline. The vector represents the local attributes of the borderline (see Figure 2).

The total value of the external force term is given by:

$$G_{ext}(P^{(t)}, C, \mathbf{I}) = \sum_{k=1}^{n} G_{score}(p_k^{(t)}, C, \mathbf{I}) \qquad (2)$$

where $P^{(t)}$ denote the polygonal snake at iteration t, and G_{score} is a correlation function that scores the extent to which a profile at a specific point $p_k^{(t)}$ matches the profile of the near by control points. We assume that the profiles change smoothly along the boundary, thus the score of $p_k^{(t)}$



(a) (b)

Figure 3. System evaluation. The user first marks four points on the boundary of the object. The system creates, in real time, an initial boundary (a). Adding one more point (b) allows the system to calculate a new and accurate boundary. With five mouse clicks, the user can cutout the object of interest.

depends on how well its profile matches $\delta(c_i)$ and $\delta(c_{i+1})$. Each of the two scores of $p_k^{(t)}$ are weighed by its arc length distance to each of nearby control points.

$$\begin{aligned} G_{score}(p_k) &= (1-\alpha)G_{match}(\delta(c_{i+1}),\delta(p_k)) \\ &+ \alpha G_{match}(\delta(c_i),\delta(p_k)) \end{aligned} \quad (3)$$

where

$$\alpha = \frac{dist(p_k, c_{i+1})}{dist(p_k, c_i) + dist(p_k, c_{i+1})}$$

G_{match} is a correlation function to grade how well any two profiles share similar image attributes:

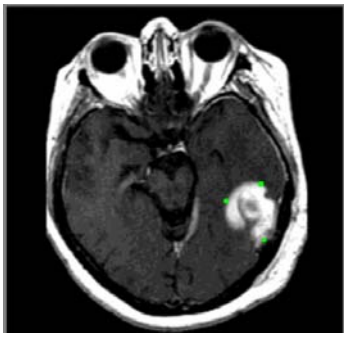
$$G_{match}(\delta(x),\delta(y)) = 1 - \frac{\sum_{l=1}^{L} abs(\delta(x_l) - \delta(y_l))}{L \cdot \Theta} \quad (4)$$

where $\delta(x)$ and $\delta(y)$ are two intensity profiles to be compared and Θ is a normalization term equal to the maximum value of the three $(\delta_i, \delta_{i+1}, \delta_k)$.

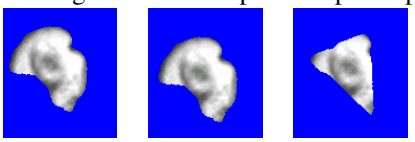
The search for the best candidate point is done along a segment defined by the outward normal at p_k. A set of S candidate points are taken along the normal. The search range is centered around the current location of p_k to allow updating its location in either way. For each candidate S_i we compute the external force score $G_{score}(S_i)$.

3.3. Internal forces

The external forces find, for each point p_k, the optimal displacement based on the profile matching out of all candidates. This is the displacement,that will maximize the match between the intensity profile of the point p_k and its two nearest control points. However, to assure smooth and locally consistent results a term in the form of internal forces is added. This smoothness term will give higher preference to candidates whose proposed displacement is similar to that of the local neighborhood. The set of neighboring points that will affect p_k is denoted by $N(p_k)$.



original image with control points super-imposed



$\lambda = 0.5$ $\lambda = 1.5$ $\lambda = 2$

Figure 4. Evaluating the importance of internal forces. We compare three identical cases, where the only difference is the weight of the internal forces (the parameter λ, in equation 1).

For each neighboring point $q_j \in N(p_k)$ we find its projection on the normal of p_k. We compute a weighted average of all these projections based on the confidence level of $G(q_j)$ in the previous iteration $t-1$. The weighted average point will be the proposed location for p_k based on the internal forces and is given by:

$$d = \frac{\sum_{j \in N_i} G(q_j^{(t-1)})(q_j^{(t-1)} - p_k^{(t-1)}) \cdot n_i^{(t-1)}}{\sum_{j \in N_i} G(q_j^{(t-1)})} \quad (5)$$

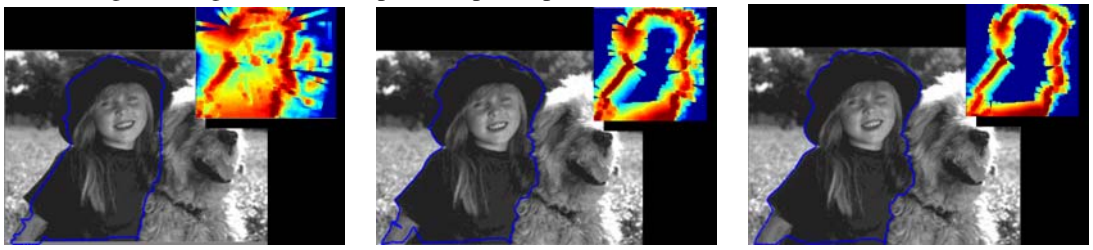
Considering all the potential candidates $\{s_1, \ldots, s_k\}$, the internal force term will prefer the one that has the minimal projection distance to the average projection d. The projection score for each candidate along the normal to p_k is given by:

$$G_{p\_score}(s_i) = 1 - \frac{|s_i - d|}{|s_0 - s_k|} \quad (6)$$

The selected candidate s_i is the one where the combination of the external and internal terms $G_{score}(s_i) + \lambda \cdot G_{p\_score}(s_i)$ is maximized.

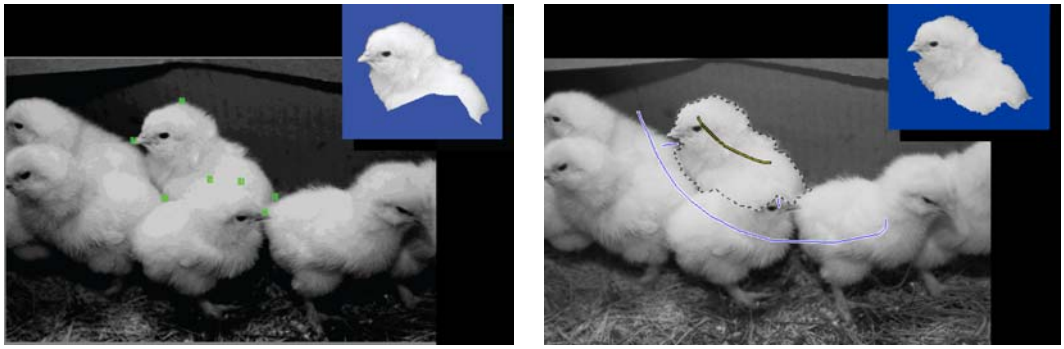
This iterative process continues until the selected candidates and thus the location of all vertices in P show consecutive displacements that are below a preset threshold. The result is a maximization of the combined functional of the boundary. Dynamic programming techniques were also considered in implementing this concept however the key novelty of unconstrained boundary definition is not sensitive to the implementation technique.

(a) Original image with control points super-imposed (b) cutout

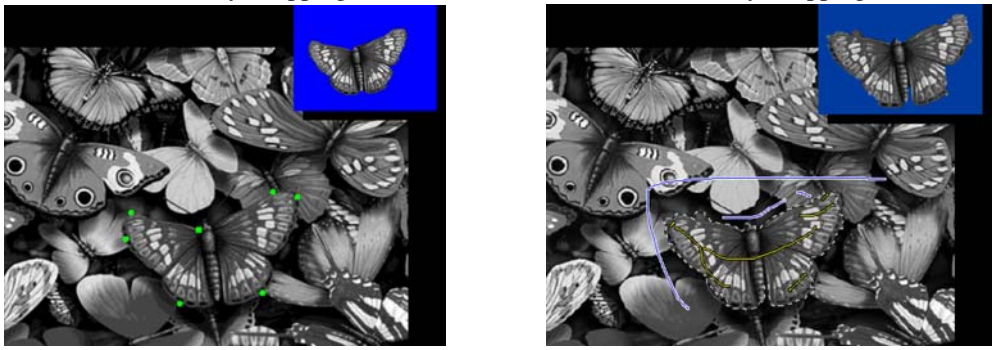


(c) Initial boundary (d) after 5 iterations (e) after 12 iterations

Figure 5. Iterations of the algorithm. The top row shows the original image (a) and the cutout (b). The bottom row (c-e) shows the iterations of the algorithm, given the control points. Within 12 iterations, the system found a boundary whose points match the control points sufficiently well. The entire process runs in real time. The attached heat maps show in color the score of candidate points along a normal in each iteration starting. Initially scores are low and wildly spread but as the polygonal snake evolves the score becomes higher ("hotter") and tightly converges to the boundary.



(a) Boundary Snapping (b) Lazy Snapping



(c) Boundary Snapping (d) Lazy Snapping

Figure 6. Comparison between Boundary Snapping and Lazy Snapping. Obtaining high quality cutouts requires a comparable amount of user interaction in both methods. However, our approach directly addresses the boundary, giving the user a direct control over the expected result.

4. Results

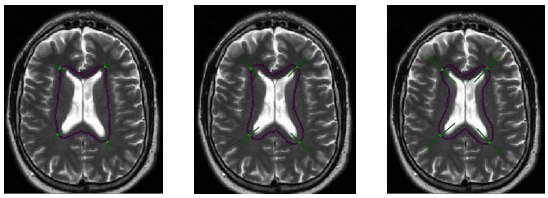
We have implemented the method, tested it on a wide variety of images and compared it to previously published techniques. In the first experiment, we show how the system is used. The user marks a small number of landmarks and a rough initial polygonal contour is created. The user can visually inspect the result and keep adding points until the outcome is satisfactory. We observed that in practice a small number of control points, quite often less than ten, are enough to cutout an object. Figure 3 shows a couple of images depicting this process. To better understand the roles of the internal and external forces, we fixed the number of landmark points and ran the algorithm multiple times, using different weights for the forces. As can be seen in Figure 4, different weights lead to different boundaries. But we found that in general a weight of around 0.7 gives satisfactory results.

Next, we evaluated the quality of our 1D appearance profile. This was done by computing a heat map that measures the similarity of different pixels to the landmarks. Figure 5 shows an example of an image and its associated heat map. As can be seen, the 1D profile used to model the boundary gives a good measure for boundary point similarity.

We demonstrate the effectiveness of the 1D intensity profile for *Margin Cutout*. In several medical applications this comes in handy. For example, in case of tumors it is often required to segment the tumor with a margin, to help in the treatment program. Figure 7 demonstrates this capability. Observe that there was no need to modify the algorithm at all.

We then compared boundary snapping to three leading techniques: lazy snapping (Figure 6), level-sets (Figure 8) and intelligent scissors (Figure 9). As can be seen, boundary snapping compares favorably with these state-of-the-art techniques. We achieve comparable results to Lazy Snapping using similar number of user input cues and applying their graph cut algorithm (without further locally editing the result). Moreover, observe in (Figure 6) (b) that there is an error in the Lazy Snapping in the boundary separating the two birds. The boundary is not obvious and it might be difficult to correct this in Lazy Snapping. It is not clear what scribble should be made to *indirectly* affect the correct estimation of the boundary. Our algorithm, on the other hand, exhibits better results, and correcting errors is straightforward and simple. The second comparison shows similar results but required additional strokes on the Lazy Snapping. In addition it can be seen that (Figure 6) (d) is still not optimal at the bottom part of the cutout.

In the comparison to level sets we achieve superior results, albeit using about twenty control points. This is clearly much more than the minimal input supplied to the level set (just a seed point). However, our method gives the user complete control and makes it easy to add control points until the results are satisfactory. In the case of Intelligent Scissors we achieve comparable results by clicking on just 11 points, as opposed to roughly tracking the entire boundary of the lamb. These comparisons demonstrate the advantages of our method and the effectiveness of our user interface.



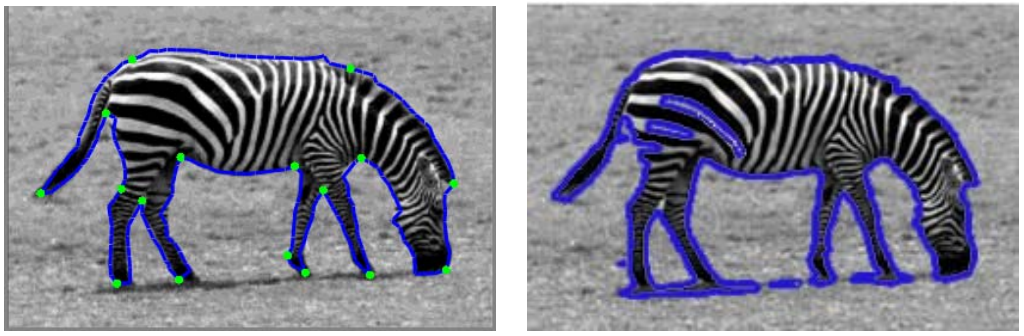
(a) $L = 10$ (b) $L = 20$ (c) $L = 30$

Figure 7. Margin Cutout: Segmenting an object with a margin. In medical applications it is often desired to cutout an object with a margin. Our Boundary Snapping automatically learns the required margin from the intensity profile of the control points. In the figures, the size of the intensity profile L was changed from 10 to 30.

Finally, Figure 10 shows results of image cutouts on a wide variety of images, including natural, outdoors images and medical images. As can be seen, a relatively small number of control points is sufficient to obtain high quality cutouts. These are challenging examples for many reasons. Figure 10 (a) has many misleading edges around the desired boundary. The texture of the objects is very diverse in color and texture. Yet, our approach restricts its search to local area around the control points and is thus not obstructed by the misleading edges in the region. Figures 10 (b-d) show that the technique is robust to somewhat vague boundaries as well. The tissue boundary on the medical image and the "furry" nature of the boundary in the kitten pose an accuracy challenge to cutout techniques. With minimal user input our technique provides a very accurate cutout due to the ability to directly control the boundary. Figures 10 (e-f) show cases in which the relationship between the object and the background is not consistent along the boundary and in some cases hardly distinguishable. However with very few control points Boundary Snapping results in accurate cutouts.

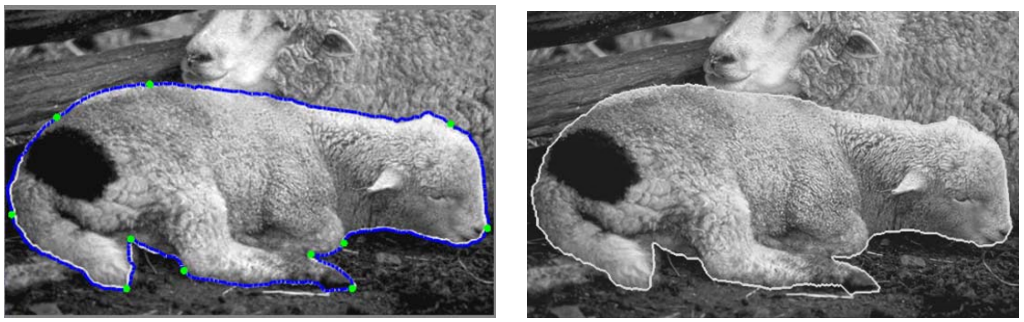
5. Conclusions

We presented Boundary Snapping, an interactive image cutout algorithm. It lets the user directly mark several points on the boundary and then attempts to match the appearance of the entire boundary with that of the user specified control points. The approach works very well in typical cases, where the foreground and background differ in appearance, as well as in challenging cases where the subject is clearly perceived, but the regions on both sides of the boundary are similar and cannot be easily discriminated. Our approach can handle these cases because it models the boundary di-



(a) Boundary Snapping (b) Level Set

Figure 8. Comparison between Boundary Snapping and Level Sets. Level set achieves impressive result, using just a single seed point. However, the result is not perfect and it is not clear how to modify the input to improve the cutout. Boundary Snapping, on the other hand, gives better results and offers a direct control of the process, albeit at a modestly higher user interaction.



(a) Boundary Snapping (b) Intelligent Scissors

Figure 9. Comparison between Boundary Snapping and Intelligent Scissors. Both methods produce similar cutouts, but differ in their interface. Intelligent Scissors requires the user to roughly track the entire contour of the object, while Boundary Snapping requires the user to click a small number of control points.

rectly, albeit, at the expense of increased user interaction. Furthermore, the boundary of the subject does not necessarily have to be the most salient or high gradient feature in the region. It gives the user full control to characterize the significant boundary and is not obstructed by misleading edges or spatial changes in the object or the background. The algorithm is fast, providing immediate visual feedback to the user, and robust, working on a variety of images, including line drawings, natural images and medical images, to name a few.

References

[1] W. A. Barrett and A. S. Cheney. Object-based image editing. In *In Proceedings of ACM SIGGRAPH*, 2002. 2

[2] Y. Boykov and M. P. Jolly. Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images. In *In Proceedings of International Conference on Computer Vision (ICCV)*, 2001. 2

[3] A. A. M. Dontcheva, M. Agarwala, S. Drucker, A. Colburn, B. Curless, D. Salesin, and M. Cohen. Interactive digital photomontage. In *In Proceedings of ACM SIGGRAPH*, 2004. 2

[4] D. Freedman and Z. Tao. Interactive graph cut based segmentation with shape priors. In *In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005. 2

[5] M. Gleicher. Image snapping. In *In Proceedings of ACM SIGGRAPH*, 1995. 2

[6] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of computer Vision*, 1(4):321–331, 1987. 3

[7] Y. Li, J. Sun, C.-K. Tang, and H.-Y. Shum. Lazy snapping. In *In Proceedings of ACM SIGGRAPH*, 2004. 2

[8] D. Martin, C. Fowlkes, and J. Malik. Learning to detect natural image boundaries using local brightness, color and texture cues. *In IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(5), 2004. 2

[9] E. N. Mortensen and W. A. Barrett. Inteliggent scissors for image composition. In *In Proceedings of ACM SIGGRAPH*, 1995. 2

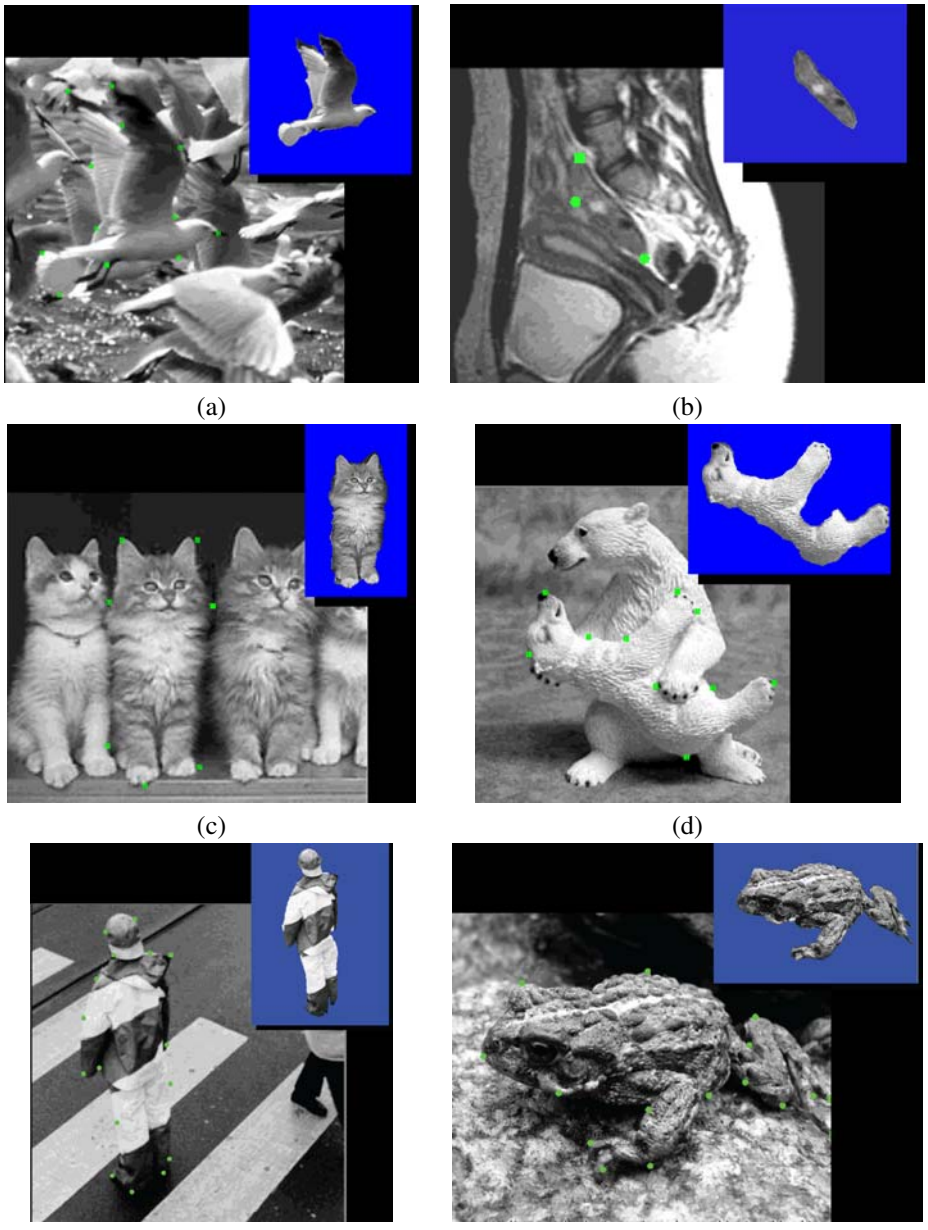


Figure 10. Results on various images. We show results of running the algorithm on various images. Observe that a small number of control points is usually sufficient to obtain a good cutout on a variety of challenging situations.

[10] E. N. Mortensen and W. A. Barrett. Toboggan-based intelligent scisors with a four parameter edge model. In *In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1999. 2

[11] S. Osher and J. A. Sethian. Fronts propagationg with curvature-dependent speed: Algorithms based on hamilto-jacobi formulations. *Journal of Computation Physics*, 79:12–49, 1988. 3

[12] P. Perez, A. Blake, and M. Gangnet. Jetstream: Probablisitic contour extraction with particles. In *In Proceedings of International Conference on Computer Vision (ICCV)*, 2001. 2

[13] L. J. Reese and W. A. Barrett. Image editing with intelligent paint. *In Proceedings of Eurographics*, 21(3):714–724, 2002. 2

[14] C. Rother, A. Blake, and V. Kolmogorov. Grabcut - interactive foreground extraction using iterated graph cuts. In *In Proceedings of ACM SIGGRAPH*, 2004. 2