Lazy Snapping

[†]Yin Li* [‡]Jian Sun [†]Chi-Keung Tang [‡]Heung-Yeung Shum
[†]Hong Kong University of Science and Technology [‡]Microsoft Research Asia

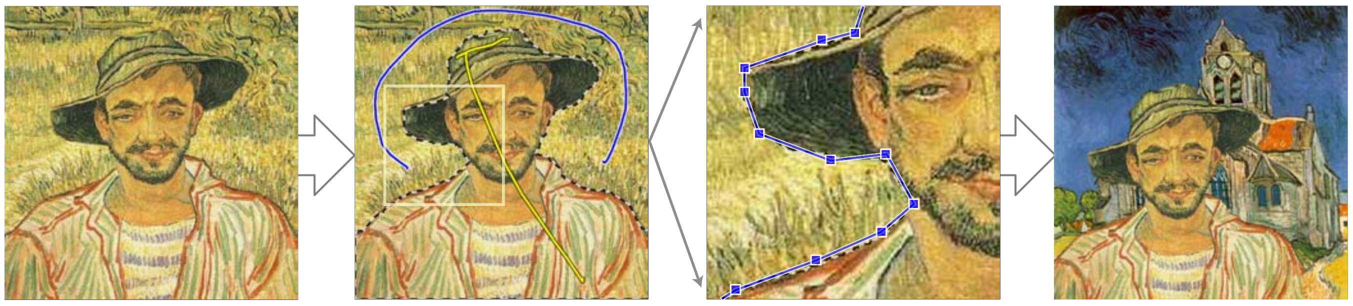

(a) Input image (b) Object Marking (c) Boundary editing (d) Output composition

Figure 1: Lazy Snapping is an interactive image cutout system, consisting of two steps: a quick object marking step and a simple boundary editing step. In (b), only 2 (yellow) lines are drawn to indicate the foreground, and another (blue) line to indicate the background. All these lines are far away from the true object boundary. In (c), an accurate boundary can be obtained by simply clicking and dragging a few polygon vertices in the zoomed-in view. In (d), the cut out is composed on another Van Gogh painting.

Abstract

In this paper, we present *Lazy Snapping*, an interactive image cutout tool. Lazy Snapping separates coarse and fine scale processing, making object specification and detailed adjustment *easy*. Moreover, Lazy Snapping provides instant visual feedback, *snapping* the cutout contour to the true object boundary efficiently despite the presence of ambiguous or low contrast edges. Instant feedback is made possible by a novel image segmentation algorithm which combines graph cut with pre-computed over-segmentation. A set of intuitive user interface (UI) tools is designed and implemented to provide flexible control and editing for the users. Usability studies indicate that Lazy Snapping provides a better user experience and produces better segmentation results than the state-of-the-art interactive image cutout tool, Magnetic Lasso in Adobe Photoshop.

Keywords: User Interface, Image Cutout, Interactive Image Segmentation, Graph Cut

1 Introduction

"Image cutout" is the technique of removing an object in a picture or photograph from its background. The cutout result is typically composited on a different background to create a new scene. Image cutout has been around for many years, and is popular in film, television, publication, and photography. It is simple enough to explain that even young children make cutouts from magazines or picture books. With the advent of digital imaging, it has become possible to specify the foreground and background on an individual pixel level, providing more accurate results than any scissors could, but no less tedious.

The task in image cutout is in specifying which parts of the image are "foreground" (the part you want to cut out) and which belong to the background. While a human finds it quite easy to specify foreground and background to another human by saying something like "cut out the tree from the field of flowers", the computer is still a long way from the sort of cognitive image understanding required to do this work unassisted. The user is forced to specify each region of foreground individually, with pixel accuracy. The tediousness of this pixel-accurate work, done in support of what is a cognitively simple task, makes image cutout a particularly frustrating task for users.

The challenge, therefore, is to come up with a way to specify the foreground that is less tedious than marking every pixel individually, without sacrificing pixel-accurate quality.

Related Work

For general image cutout, there are two main methods that improve on standard pixel-level selection tools: boundary-based and region-based. Each of these methods takes features of the image that the computer can detect (such as color consistency) and uses them to help automate or guide the foreground specification process.

Boundary-based methods cut out the foreground by allowing the user to surround its boundary with an evolving curve. The user traces along the object boundary and the system optimizes the curve in a piecewise manner. Examples include intelligent scissor [Mortensen and Barrett 1995; Mortensen and Barrett 1999], image snapping [Gleicher 1995] and Jetstream [Perez and Blake 2001].

While easier than just selecting pixels manually with a traditional selection tool, these techniques still demand a large amount of attention from the user. There is never a perfect match between the features used by the algorithms and the foreground image. As a result, the user must control the curve carefully. If a mistake is made,

*This research was done when Yin Li was with Microsoft Research Asia as an intern.

the user has to "back up" the curve and try again. The user is also required to enclose the entire boundary, which can take some time for a complex, high-resolution object. The close control required interferes with the user's ability to get an overview of their progress. It is difficult to zoom in and out of the image while you are dragging the pixel-accurate boundary line. Finally, once the boundary is specified, the tool is no longer helpful. Any errors must be cleaned up at the end using traditional selection tools (e.g., using the Lasso tool with Boolean operation in Photoshop).

Recently, researchers have managed to improve image cutout by using region-based methods, e.g., magic wand in Photoshop, intelligent paint [Reese and Barrett 2002; Barrett and Cheney 2002], marker drawing [Falcao et al. 2000], sketch-based interaction [Tan and Ahuja 2001], interactive graph cut image segmentation [Boykov and Jolly 2001], GrabCut [Rother et al. 2004]) and interactive image Photomontage [Agarwala et al. 2004]. Region-based methods work by allowing the user to give loose hints as to which parts of the image are foreground or background without enclosing regions or being pixel accurate. These hints usually take the form of clicking or dragging on foreground or background elements, and are thus quick and easy to do. An underlying optimization algorithm extracts the actual object boundary based on the user input hints.

Region-based methods allow the user to operate at whatever scale they want. They also show partial results. After each hint, the foreground/background specification becomes more and more accurate. The problem with region-based techniques is that there are often cases where the features used by the region detection algorithms do not match up with the desired foreground or background. Areas in shadow, low-contrast edges, and other ambiguous areas can be extremely tedious to hint. Sometimes, they cannot be hinted and need to be specified explicitly by hand.

Clearly, there is still a need for a user interface that can combine the quick hinting of region-based approaches while still providing a simple affordance for pixel-accurate boundary editing.

Our approach

We propose Lazy Snapping, which is a novel coarse-to-fine UI design for image cutout. As shown in Figure 1, Lazy Snapping consists of two steps: a quick *object marking* step (b) and a simple *boundary editing* step (c). The first step, object marking, works at a coarse scale, which specifies the object of interest by a few marking lines (Section 2). The second step, boundary editing, works at a finer scale or on the zoomed-in image, which allows the user to edit the object boundary by simply clicking and dragging polygon vertices (Section 3).

Our system inherits the advantages of region-based and boundary-based methods in two steps. The first step is intuitive and quick for object context specification, while the second step is easy and efficient for accurate boundary control.

Inspired by [Boykov and Jolly 2001], we also formulate image cutout as a graph cut problem in both steps. Furthermore, at the object marking step, we propose an efficient graph cut algorithm by employing pre-computed over-segmentation so that the marking UI can provide instant visual feedback for users. At the boundary editing step, we introduce a simple polygon editing UI, and use the polygon locations as soft constraints to improve snapping results around ambiguous or low contrast edges.

We have conducted usability studies (Section 4) to compare Lazy Snapping with the state-of-the-art interactive cutout tool, Magnetic Lasso in Photoshop, which has perhaps the best implementation of intelligent scissor. It shows that Lazy Snapping outperforms in terms of ease of use, efficiency, and quality of results. We have experimented with our system on many natural images.

2 Object Marking

In the object marking step, the major task is to allow the user to conceptually group the foreground object against its background. Instead of tracing the object boundary, our system allows users to use lines and curves to specify the extent of the object of interest.

2.1 UI Design

To specify an object, a user marks a few lines on the image by dragging the mouse cursor while holding a button (left button indicating the foreground, and right button for the background). A yellow line or a blue line is displayed for the foreground marker or background marker respectively. This high level, painting-type UI does not require very precise user inputs. As shown in Figure 1(b), most marking lines are in fact far from the object boundary. Similar marking UI to separate object from background is also presented in[Falcao et al. 2000; Boykov and Jolly 2001; Fails and Olsen 2003] for image segmentation or gesture tracking for camera-based interaction.

The segmentation process is triggered once the user releases the mouse button after each marking line is drawn. The user then inspects the segmentation result on screen and decides if more lines need to be marked. It is therefore critical that our system generates the cutout boundary with very little delay. Our system adopts a novel interactive graph cut algorithm to optimize the object boundary, by maximizing both the color similarity inside the object and the gradient along the boundary.

2.2 Graph Cut Image Segmentation

An image cutout problem can be posed as a binary *labelling* problem. Suppose that the image is a graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$, where $\mathcal{V}$ is the set of all nodes and $\mathcal{E}$ is the set of all arcs connecting adjacent nodes. Usually, the nodes are pixels on the image and the arcs are adjacency relationships with four or eight connections between neighboring pixels. The labelling problem is to assign a unique label x_i for each node $i \in \mathcal{V}$, i.e. $x_i \in \{\text{foreground}(=1), \text{background}(=0)\}$. The solution $X = \{x_i\}$ can be obtained by minimizing a Gibbs energy $E(X)$ [Geman and Geman. 1984]:

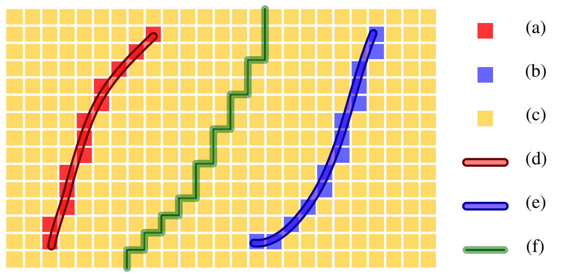
$$E(X) = \sum_{i \in \mathcal{V}} E_1(x_i) + \lambda \sum_{(i,j) \in \mathcal{E}} E_2(x_i, x_j) \qquad (1)$$

where $E_1(x_i)$ is the likelihood energy, encoding the cost when the label of node i is x_i, and $E_2(x_i, x_j)$ is the prior energy, denoting the cost when the labels of adjacent nodes i and j are x_i and x_j respectively.

In this paper, we will concentrate on how to define the energy terms E_1 and E_2 according to user input. We refer readers to [Boykov and Jolly 2001] for a detailed formulation of energy minimization as a graph cut problem and how to solve it. The graph cut algorithm has also been used in the computer graphics community, such as Graph Cut Textures [Kwatra et al. 2003], GrabCut [Rother et al. 2004] and Photomontage [Agarwala et al. 2004].

Once the user marks the image, two sets of pixels intersecting with the foreground and background markers are defined as *foreground seeds* $\mathcal{F}$ and *background seeds* $\mathcal{B}$ respectively, as shown in Figure 2.

Likelihood energy. In Equation (1), E_1 encodes the color similarity of a node, indicating if it belongs to the foreground or background. To compute E_1, first the colors in seeds $\mathcal{F}$ and $\mathcal{B}$ are clustered by the K-means method [Duda et al. 2000]. The mean colors of the foreground and background clusters are denoted as $\{K_n^{\mathcal{F}}\}$ and $\{K_m^{\mathcal{B}}\}$ respectively. The K-means method is initialized to have 64 clusters in our experiments. Then, for each node i, we compute the minimum distance from its color $C(i)$ to foreground clusters as



(a) Foreground seeds $\mathcal{F}$ (b)Background seeds $\mathcal{B}$ (c)Uncertain regions $\mathcal{U}$
(d) Foreground marker (e) Background marker (f) Graph cut result

Figure 2: Graph cut formulation for Object Marking. The graph cut algorithm is defined on $\mathcal{F}$, $\mathcal{B}$, and $\mathcal{U}$. All these nodes participate in the optimization process and are assigned a unique label, either foreground or background.

$d_i^{\mathcal{F}} = \min_n \|C(i) - K_n^{\mathcal{F}}\|$, and similarly $d_i^{\mathcal{B}} = \min_m \|C(i) - K_m^{\mathcal{B}}\|$. Therefore, $E_1(x_i)$ is defined as follows:

$$\begin{cases} E_1(x_i = 1) = \quad 0 & E_1(x_i = 0) = \quad \infty & \forall i \in \mathcal{F} \\ E_1(x_i = 1) = \quad \infty & E_1(x_i = 0) = \quad 0 & \forall i \in \mathcal{B} \\ E_1(x_i = 1) = \frac{d_i^{\mathcal{F}}}{d_i^{\mathcal{F}} + d_i^{\mathcal{B}}} & E_1(x_i = 0) = \frac{d_i^{\mathcal{B}}}{d_i^{\mathcal{F}} + d_i^{\mathcal{B}}} & \forall i \in \mathcal{U} \end{cases} \quad (2)$$

Here, $\mathcal{U} = \mathcal{V} \setminus \{\mathcal{F} \cup \mathcal{B}\}$ is the uncertain region (Figure 2). The first two equations guarantee that the nodes in $\mathcal{F}$ or $\mathcal{B}$ will always have the label consistent with user inputs. The third equation encourages the nodes to have the label with similar colors to foreground or background.

Prior energy. We use E_2 to represent the energy due to the gradient along the object boundary. We define E_2 as a function of the color gradient between two nodes i and j:

$$E_2(x_i, x_j) = |x_i - x_j| \cdot g(C_{ij}) \quad (3)$$

where $g(\xi) = \frac{1}{\xi+1}$, and $C_{ij} = ||C(i) - C(j)||^2$ is the $L2$-Norm of the RGB color difference of two pixels i and j. Note that $|x_i - x_j|$ allows us to capture the gradient information only along the segmentation boundary. In other words, E_2 is a penalty term when adjacent nodes are assigned with different labels. The more similar the colors of the two nodes are, the larger E_2 is, and thus the less likely the edge is on the object boundary.

To minimize the energy $E(X)$ in Equation (1), we use the max-flow algorithm in [Boykov and Kolmogorov 2001]. This algorithm is specially designed for some vision problems. Unfortunately, as shown in the last column of Table 1, it fails to provide interactive visual feedback for real life image cutouts.

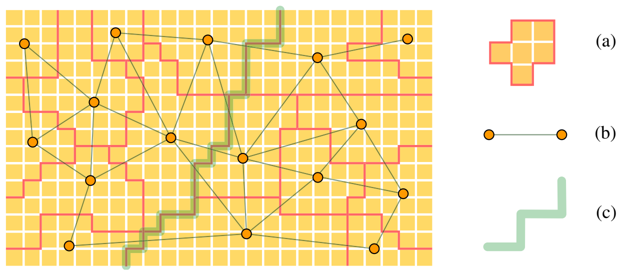
Image	Dimension	Nodes Ratio	Edges Ratio	Lag with Pre-segmentation	Lag without Pre-segmentation
Boy	$(408, 600)$	10.7	16.8	0.12s	0.57s
Ballet	$(440, 800)$	11.4	18.3	0.21s	1.39s
Twins	$(1024, 768)$	20.7	32.5	0.25s	1.82s
Girl	$(768, 1147)$	23.8	37.6	0.22s	2.49s
Grandpa	$(1147, 768)$	19.3	30.5	0.22s	3.56s

• The nodes (edges) ratio is the number of pixels (connection between pixels) divided by the number of nodes (edges) after the pre-segmentation.

• The feedback lag is the delay from when the user releases the mouse to when the object boundary is displayed.

• All lags are timed on a laptop PC with Centrino 1.5GHz CPU and 512M memory.

Table 1: Performance comparison of the graph cut segmentation algorithms with and without pre-segmentation on the images shown in Figure 9.



(a) A small region by the pre-segmentation. (b) The nodes and edges for the graph cut algorithm with pre-segmentation. (c) The boundary output by the graph cut segmentation.

Figure 3: Our new graph cut algorithm works on the graph whose nodes are small regions from watershed segmentation.

2.3 Graph Cut with Pre-segmentation

To improve efficiency, we introduce a novel graph cut formulation which is built on a pre-computed image over-segmentation, instead of image pixels. We choose the watershed algorithm [Vincent and Soille 1991], which locates boundaries well, and preserves small differences inside each small region.

We again formulate object cutout as a graph cut problem where the nodes are instead the segmented regions from the watershed segmentation. As shown in Figure 3, we use the same notation $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ for the new graph, while the nodes $\mathcal{V}$ are the set of all small regions from pre-segmentation, and the edges $\mathcal{E}$ are the set of all arcs connecting adjacent regions.

The foreground seeds $\mathcal{F}$, the background seeds $\mathcal{B}$, and the uncertain region $\mathcal{U}$ are defined similarly as in Section 2.2, except that now these nodes are small regions instead of pixels. The likelihood energy E_1 is also similar to Equation (2) while the color $C(i)$ is computed as the mean color of the small region i .

For the prior energy E_2 in Equation (3), we compared two definitions of C_{ij}: 1) C_{ij} is the mean color difference between the two regions i and j; 2) Similarly defined C_{ij}, but it is weighted by the shared boundary length between regions i and j. In our experiments, similar results were obtained.

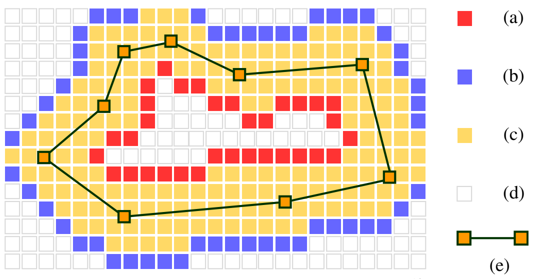
Since watershed segmentation provides a good super set of object boundaries, this approximation produces reasonable results and improves the speed significantly. As shown in Table 1, the number of nodes and edges for the graph cut algorithm is reduced by more than 10 times compared to the pixel based method in our experiments with real life images. Most importantly, our new algorithm is able to feedback the cut out results almost instantly.

3 Boundary Editing

Although the object marking step preserves the object boundary as accurately as possible, there still exist some errors, especially around ambiguous and low contrast edge boundaries. Therefore, we design a simple polygon editing UI for the user to refine the object boundary.

3.1 UI Design

The object boundary produced from the previous step is first converted into editable polygons. The polygon is constructed in an iterative way: the initial polygon has only one vertex, which is the point with the highest curvature on the boundary. At each step, we compute the distance of each point on the boundary to the polygon in the previous step. The farthest point is inserted to generate a new polygon. The iteration stops when the largest distance is less than a pre-defined threshold (typically 3.2 pixels).



(a) Foreground seeds $\mathcal{F}$ (b) Background seeds $\mathcal{B}$
(c) Uncertain regions $\mathcal{U}$ (d) Pixels ignored by graph cut
(e) Polygon vertices and lines

Figure 4: Graph cut formulation for boundary editing. Only pixels in $\mathcal{F}$, $\mathcal{B}$, or $\mathcal{U}$ are considered in optimization. The polygon location is encoded as an energy term to guide the optimization to snap to user inputs.

Two UI tools are provided for polygon editing:

Direct vertex editing allows users to drag the vertex to adjust the shape of the polygon. Users can add or delete vertices as well. Multiple vertices can be grouped and processed together.

Overriding brush enables users to draw a single stroke to replace a segment of a polygon. This is more efficient than dragging many vertices individually.

The overriding brush is inspired by the Paintbrush tool in Adobe Illustrator. The user brushes a stroke starting and stopping at two points A and B on the original polygon so that the original polygon is split into two parts, one of which has less angle difference to the user stroke. This part is replaced by the user stroke to generate a new polygon. The angle of the user stroke and the two parts of the polygon is measured by the tangent direction at point A and from A to B.

Once the user releases the mouse button after each polygon editing operation, the system will optimize the object boundary using the graph cut segmentation algorithm again. The optimized boundary automatically snaps to the object boundary even though the polygon vertices may not be on it. Compared with a simple polygon boundary where the user needs to modify so many vertices, our UI uses many fewer polygon vertices to describe the object shape.

3.2 Boundary Editing using Graph Cut

Again, we formulate boundary editing as a pixel-based graph cut problem in a small band around the polygons. The band is 7 pixels wide by default. Figure 4 shows foreground seeds $\mathcal{F}$, background seeds $\mathcal{B}$ and uncertain region $\mathcal{U}$. Given the editable polygon, $\mathcal{U}$ is a band computed by dilating the polygon, whereas $\mathcal{F}$ and $\mathcal{B}$ are defined as the inner and outer boundaries of $\mathcal{U}$ respectively.

The likelihood energy E_1 is defined as in Equation (2) in the object marking step. But the prior energy E_2 is defined differently. In addition to the gradient term, E_2 uses the polygon locations as soft constraints, in order to deal with ambiguous and low contrast gradient boundaries:

$$E_2(x_i, x_j) = |x_i - x_j| \cdot g\left((1-\beta)\cdot C_{ij} + \beta \cdot \eta \cdot g(D_{ij}^2)\right) \quad (4)$$

where $g(\cdot)$ is the same as in Equation (3), D_{ij} is the distance from the center of arc (i, j) to the polygon and η is the scale to unify the units of the two terms (typical value is 10).

In Equation (4), $\beta \in [0, 1]$ is used to control the influence of $D(i, j)$. A typical value of β is 0.5 and it works well in most of our experiments, although we allow expert users to adjust this parameter for better performance. Note that $\beta = 1$ makes the graph cut segmentation output the result that is snapped onto the polygon, regardless of the image gradient.

When color gradient C_{ij} is small, $g(D_{ij}^2)$ dominates E_2, which encourages the result to snap close to the polygon location. This is shown in Figure 5 where low contrast edges are very difficult to snap without polygon soft constraints. As shown in Figure 5(a), it is also a difficult example for region-based methods (e.g., in the object marking step).

If there are two edges with comparable strength, the polygon location can also help users to select the desired one, as shown in Figure 6(b). Otherwise, the segmentation result may not be fully controlled by the polygon, as shown in Figure 6(c).

Hard vertex constraint: The users may prefer to specify manually a polygon vertex to be a "hard" constraint, so that the system ensures the graph cut segmentation result to pass through this vertex. For this hard constrained vertex, the uncertain region $\mathcal{U}$ is automatically split into two parts along its bisector. The two "split" lines are added into foreground seeds $\mathcal{F}$ and background seeds $\mathcal{B}$ respectively, so that graph cut segmentation must output a result passing through this vertex, because it is the only connection between the foreground and background at this place.

4 Usability Study

We believe that Lazy Snapping is superior to existing cutout methods in being easier to learn and able to produce results of equal or better quality in less time. In order to test this, we have conducted a usability study that compared the performance of our Lazy Snapping prototype system to Magnetic Lasso, Adobe Photoshop's image cutout tool.

Methodology

Fourteen subjects were selected. Ten were novices with little or no experience with Photoshop or its image cutout tools, while four were Photoshop experts. Each subject was given a five minute

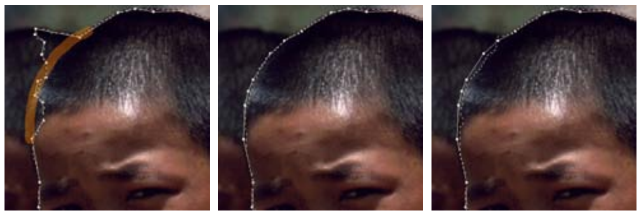


Figure 5: The polygon soft constraint can override edge locations at low contrast regions. (a) The object marking step produces a bad boundary. Using the polygon overriding brush (thick orange line) can replace a segment of polygon (b) Enabling the polygon as a soft constraint, the result (dotted line) is very close to the polygon (solid line). (c) Otherwise, the optimization is vulnerable to noise due to weak edges.

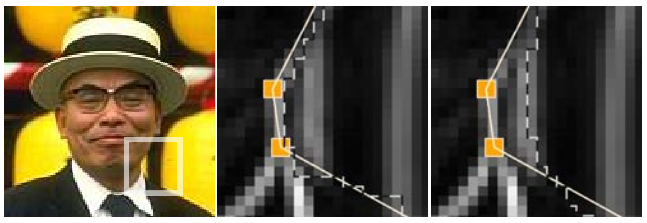


Figure 6: (a) is the original image, and (b), (c) show the color gradient image of the region marked on (a) as a white square. (b) With polygon soft constraints, users can select which strong edge to snap. (c) Without polygon soft constraints, the same input polygon may produce erroneous edges because of the inherent edge ambiguity.

instruction session on the Lazy Snapping software. The novices were also given five minutes of instruction on Photoshop's magnetic lasso tool. All users were allowed to experiment with both software packages until they were comfortable that they understood their functions.

The study consisted of two tasks. In the first task, the subjects had to cut out 4 images (A, B, C and D in Figure 7) as accurately as possible. The subjects were asked to work as quickly as they could without sacrificing accuracy. For each image, the subject had access to a printed version of the desired cutout. After they completed the task using one software package, they then repeated the task with the same 4 images using the other software. The order was alternated between subjects in case there was an ordering effect, with half of the subjects using Photoshop first, the other half using Lazy Snapping. For the second task, the subjects were given another 4 images (E, F, G and H in Figure 7) to cut out, but this time they were given only 60 seconds per image. They were instructed to get the cutout as accurately as possible in the allotted time. Again, the order was alternated between subjects.

When using Photoshop, the users were advised to use magnetic lasso as the major tool, but were also allowed to use other tools in Photoshop, such as free lasso, and work path editing, etc.

Subjects were videotaped and their cutout results were saved for detailed logging and quality analysis.

We evaluate the quality of cutouts by measuring the number of error pixels in the images. Avoiding bias, we compute the quality by averaging the number of error pixels for four 'ground truth' cutouts, which are produced by two experts (not selected as subjects) using Lazy Snapping and Photoshop respectively. We also exclude the pixels in the hairy and furry regions, to avoid the influence of subjective recognition.

Ease of Use

We tested ease of use by counting the number of errors made by the novice users. Using the video tapes, we counted the number of times the users chose the incorrect tool or had to invoke the UNDO command. For example, while the user wants to draw foreground in Lazy Snapping, the background brush is used instead; when using the magnetic lasso, the user clicks the zoom button on the navigator tool window, which will produce an unexpected result. We found the error rate of Lazy Snapping to be less than 20% of the rate of Photoshop on the same image. Users also subjectively reported Lazy Snapping to be far easier to use than Photoshop.

Better Quality in Less Time

We tested time improvements by measuring how long it took users to complete the first task. We found subjects using Lazy Snapping overall took less than 60% of the time than they did when using Photoshop (Figure 8(a)(b)). The exact benefit varied widely depending on the subject and the image (standard deviation is 30%).

Figure 7: Images used in usability study. The four images in the first row (A, B, C, and D) are for the first task. And the other four (E, F, G, and H) are for the second task.

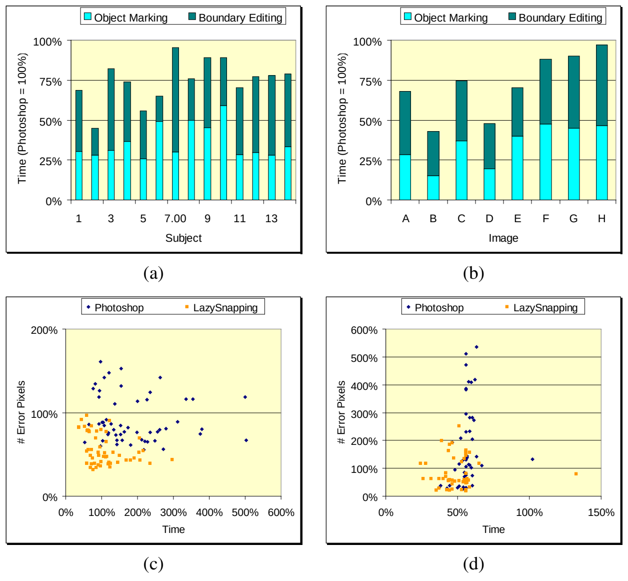


Figure 8: (a) and (b) illustrate the average time of cutout process across fourteen subjects and eight images respectively. We normalize the time by that of Photoshop for each column, so that all data can be compared together. Moreover, the number of error pixels to the time is shown in (c) for the first task and in (d) for the second task. We normalized the time and quality by the mean of all samples of each image for all subjects, so that the data from different images can be aligned around 100% for comparison. Lazy Snapping is clustered at the lower left corner, indicating better quality in less time.

We compared the quality of the Photoshop result with that of the Lazy Snapping result. For the second task, Lazy Snapping has less than 60% (the average of all 14 subjects and 4 images) the number of error pixels than Photoshop has. In this time-restricted task, Lazy Snapping was a clear winner. Most subjects were able to complete the entire task in the 60 seconds allotted (86% less time than Photoshop), and for those that were not, Lazy Snapping produced satisfactory intermediate results (less than 53% error pixels than Photoshop). Photoshop's magnetic lasso tool does not produce intermediate results. Users who ran out of time were left with large errors. (See Figure 8(d)).

Subject Feedback

Overall, subjects preferred Lazy Snapping to the tools in Photoshop. They reported it to be "much easier" and "almost magic". One expert user expressed a concern that the ease of working with Lazy Snapping might encourage him to be lazy himself and perform less accurately than with the more tedious traditional tools. Other users made suggestions for combining the Lazy Snapping tools with existing tools like the lasso and magic wand. Several users expressed some dissatisfaction with the two steps and wondered if we could make it easier to go back and forth between them. We are considering these and other suggestions for improving the user experience.

5 Experiments and Summary

Figure 9 shows more examples produced by Lazy Snapping. The number of object markers and times for polygon editing are also listed for each image.

We use Coherent matting [Shum et al. 2004], an extended Bayesian

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | sales@docketalarm.com | 1-866-77-FASTCASE

fastcase® Smarter legal research.