ClickRemoval: Interactive Pinpoint Image Object Removal

Frank Nielsen
Sony Computer Science Laboratories, Inc
Tokyo, Japan
Frank.Nielsen@acm.org

Richard Nock
Antilles-Guyane University
Martinique, France
Richard.Nock@martinique.univ-ag.fr

ABSTRACT

In this paper, we explore the problem of deleting objects in still pictures. We present an interactive system based on an intuitive user-friendly interface for removing undesirable objects in digital pictures. To erase an object in an image, a user indicates which object is to be removed by simply pinpointing it with the mouse cursor. As the mouse cursor rolls over the image, the current implicit selected object's border is highlighted, providing a visual feedback. In case where the computer-segmented area does not match the users' perception of the object, users can further provide a few inside/outside object cues by clicking on a small number of object or nonobject pixels. A small number of such cues is generally enough to reach a correct matching, even for complex textured images. Afterwards, the user removes the object by clicking the left mouse button, and a hole-filling technique is initiated to generate a seamless background portion. Our image manipulation system consists of two components: (i) fully automatic or partially user-steered image segmentation based on an improved fast statistical region-growing segmentation, and (ii) texture synthesis or image inpainting of irregular shaped hole regions. Experiments on a variety of photographs display the ability of the system to handle complex scenes with highly textured objects.

Categories and Subject Descriptors

I.3.6 [**Computer Graphics**]: Methodology and Techniques—Interaction Techniques; I.4.6 [**Image Processing and Computer Vision**]: Segmentation—Pixel classification; partitioning

General Terms

Algorithms

Keywords

User interface, computational photography, user-steered segmentation, inpainting, texture synthesis

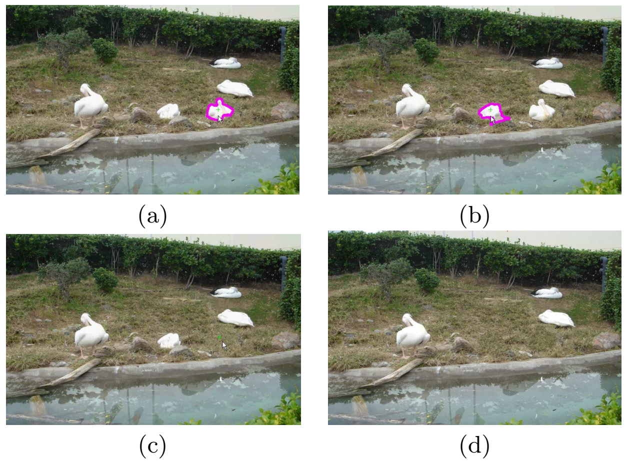

Figure 1: ClickRemoval is an interactive image object removal system. As the user's mouse cursor rolls over the image, the current implicitly selected object is highlighted, (a) and (b). By simple mouse clickings objects are removed and background portions are instantaneously synthesized, (c) and (d).

1. INTRODUCTION

Removing and cutting objects from digital pictures are main operations of desktop publishing (DTP) for which many dedicated tools have been designed and refined over the years (e.g., the *magnetic lasso* or the *magic wand* of Adobe® Photoshop®). Image cutouts are typically pasted (composited) on a different background for photomontages. Removing objects is important in the movie industry to obtain clean plates (say, remove camera tripods or other calibration materials that have been used on stage to facilitate postprocessing computer graphics effects). Clean plates are usually obtained by synthesizing the *background mosaic* by tracking and registering a sequence of frames and removing objects by manually painting them for each frame. With the advent of image inpainting [2] and texture synthesis technologies for generating seamlessly image portions, removing objects in still pictures becomes today an essential primitive of image retouching software. To remove objects in photographs (Figure 1), we do not need a pixel accuracy that requires to pull out the alpha matte but rather requires a *coarse* bounding region that separates the selected object from the remaining background of the scene. Because object cognition by computers is still very far from human abilities, the challenge consists in designing an intuitive user interface (UI) and corresponding effective user-steered segmentation algorithm for selecting objects by putting the user's high-level cognition in the loop.

1.1 Related work

Prior object selection work in images are classified into two categories:

Contour-based selection. A user marks the object boundaries by coarsely and piecewisely sketching with the mouse cursor the contours that are *on-the-fly* finely optimized to fit the object boundaries. The first contour-based methods were developed independently in 1995 using either dynamic programming [7] (*intelligent scissors*) or gradient descent optimization [4] (*image snapping*). Those methods are also better known today as *magnetic lassos*. Intelligent scissor techniques were later refined in [8]. Recently, a Monte-Carlo probabilistic system, called *jetstream* [10], has been designed to extract contours using particles. Contour-based selection work particularly well for in-focus objects out-of-focus background images, where the objects consist of a single outer contour. For more complex topology objects such as grid-like objects containing many holes, those methods are time consuming as they require users to trace each inner contour (Figure 6).

Region-based selection. A user gives a few hints on which portion of the image is the object and which image pixels are the background, and an optimization algorithm then extracts the object based on these cues. The *magic wand* tool of Photoshop is such a typical system. Other approches are either based on segmentation and triangulation [3, 1, 14] or graph cuts [12, 5].

Note that even for image cutouts, a coarse extraction is often enough as it allows to initialize a trimap (labeling of images into background/object/undefined areas) to precisely extract objects with alpha mattes [13, 6] as a postprocessing operation. Matte extraction is required for copy-pasting translucent or furry objects that have potentially soft pixel memberships (pixels being a mixture of object/background colors).

1.2 System overview

Our system ClickRemoval consists of two basic modules: (1) user-steered segmentation, and (2) hole-filling. In the ideal scenario, the user just has to pinpoint objects s/he wants to remove and press the left mouse button to remove and synthesize *instantaneously* the background part (see Figure 1). Because automatic segmentation may yield not expected results, we provide a simple mechanism to input bias by clicking just a few object/background pixels (see Figure 2). The next section describes the ClickRemoval user interface. Section 3 present the novel statistical region-growing algorithm, and Section 4 concludes the paper.

2. USER INTERFACE DESIGN

The user starts by loading a picture and moves the mouse over the image (Figure 3.(1-3)a). Whenever the mouse moves, the segmented area implicitly defined by the cursor position is highlighted in real-time (Figure 3.(1-3)b). Computing segmentations is done in almost linear-time, as described in Section 3. Once an object is removed by the user, a fast hole-filling texture synthesis procedure is triggered [15] (Figure 3.1c, 3.2d and 3.3c).

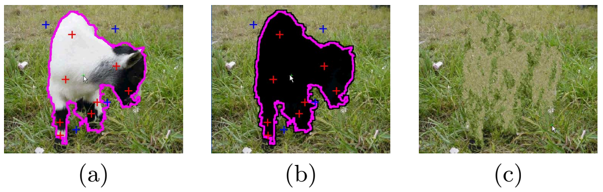

Figure 2: Matching computer/human object definitions by putting the user in the loop and iteratively refining manually the segmentation. (a) depicts the segmentation obtained after the user manually pinpointed object (red) and background (blue) cues. (b) is the hole created by removing the slightly dilated object, and (c) is the result image after texture synthesis.

ClickRemoval has three operation modes:

1. To remove the implicitly selected area Fig 3.(1b), the user clicks the mouse left button. The system computes a bounding box around the removed area (expands it by some factor) and initialize texture synthesis using the remaining pixels inside the box (Figure 3.1.(1c)). We implemented the per-pixel texture synthesis of Wei and Levoy [15] because of its flexibility and real-time performance in our setting (see also [2]).

2. Sometimes, we prefer to *design* how to fill the hole using another part of the image. We then scribble the image by pushing the left[1] mouse button and moving the mouse cursor over the portion of the image we are interested to initialize the texture synthesis (Figure 3.(2c)). We define the selected pixels by taking either the bounding box of the stroke or choosing the image pixels falling within some prescribed distance to the stroke.

3. Automatic segmentation may fail to deliver appropriate object decompositions. In case of failure, the user presses the SHIFT key and the mouse left (object) or right (background) mouse to indicate prior cues (Figure 2 and Figure 3.(3b)). ClickRemoval then refines the segmentation to satisfy the constraints of the user's cues. Texture synthesis is either initialized automatically the remaining pixels of an enlarged bounding box, or user-steered (as described in 2.).

3. STATISTICAL IMAGE SEGMENTATION

Image segmentation is computed using a fast iterative statistical region-growing process. The region-growing segmentation framework dates back to the late 60s. Since then, it has been a very popular method of image processing/computer vision [11]. Region growing starts by initializing to each pixel a corresponding single-pixel region (say, pixel region $\mathbf{R}_l = \{(x, y)\}$ has initially ID number $l = x+yw$, where w denotes the image width). At each iteration of the region growing algorithm, we consider the region adjacency graph (RAG), and based on a merging predicate, decide to merge or not the pair of adjacent regions that is under consideration. Usually the RAG is dynamically updated. The segmentation result strongly depends on: (1) the merging

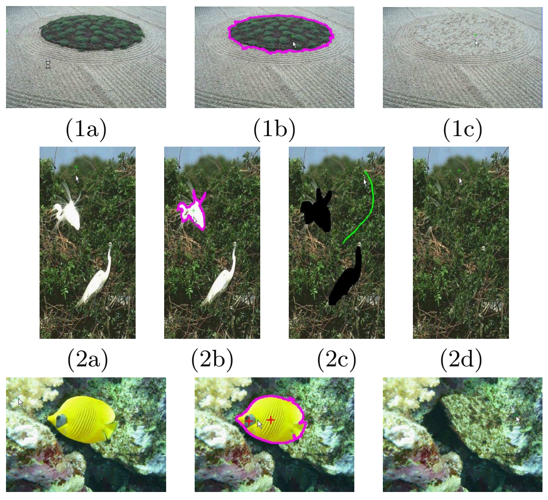

Figure 3: The three different modes of the ClickRemoval interface: fully automatic, manual texture selection, and both user-steered segmentation/texture selection.

predicate, and (2) the order in which pairs of adjacent regions are inspected. Let $\mathbf{R}_i$ and $\mathbf{R}_j$ denote a pair of adjacent regions. A typical merging predicate $P(\mathbf{R}_i, \mathbf{R}_j)$ is to test whether $|\overline{\mathbf{R}_i} - \overline{\mathbf{R}_j}| \leq \max\{\sigma_i, \sigma_j\}k$, where $\overline{\mathbf{R}_i}, \sigma_i$ (resp. $\overline{\mathbf{R}_j}, \sigma_j$) is the color mean and variance of region $\mathbf{R}_i$ (resp. $\mathbf{R}_j$), and k is a prescribed constant. Let $|\mathbf{R}|$ denote the number of pixels defining region $\mathbf{R}$. Let $\mathbf{R}(p)$ denote the region containing pixel p. In our recent segmentation work [9] (2005), we developed a fast linear-time algorithm with provably guaranteed segmentation bound based on a statistical image generation model. We presented: (1) a concentration inequality based on statistical aggregation phenomena[2] and showed that in practice it is enough to consider (2) a *static order* of region pairs. Because regions are disjoint sets of pixels, we can easily merge regions and retrieve their IDs (provided pixel handles) using the optimal union-find data-structure of Tarjan. The algorithm is further shown robust to noise and handle occlusions (that is, the method can segment as a *single object* several connected areas of the image that belong to the same object). We summarize the static order region-growing segmentation algorithm:

```
RegionMerging($\mathbf{I}$)
1. ◁ 4-connectivity of pixels $C_4$ ▷
2. $\mathbf{P} \leftarrow \{(p_l, q_l, C_l = |\mathbf{I}(q_l) - \mathbf{I}(p_l)|) \mid \text{with } q_l \in C_4(p_l)\}$
3. ◁ Sort $\mathbf{P}$ in increasing order according to key $C_l$ ▷
4. Sort($\mathbf{P}$)
5. for $i \leftarrow 1$ to $|\mathbf{P}|$
6.   do
7.      if FindRegionID($p_l$) $\neq$ FindRegionID($q_l$)
8.        then if MergePredicate($\mathbf{R}(p_l), \mathbf{R}(q_l)$)
9.               then MergeRegions($\mathbf{R}(p_l), (q_l)$)
```

[2]For example, the sum of independent uniform random variables yields a Gaussian random variable (central limit theorem). More generally, statistical aggregation phenomena have been recently found for random variables satisfying loose distribution assumptions [9].

The union-find data-structure is implemented as follows:

```
InitializeRegionID($x$)
1. parent($x$) $\leftarrow x$
2. rank($x$) $\leftarrow 0$
FindRegionID($x$)
1. ◁ Walk from $x$ to the leader pixel element ▷
2. while $x \neq$ parent($x$)
3.   do $x \leftarrow$ parent($x$)
4. return $x$
MergeRegions($x, y$)
1. ◁ Union by rank and path compression ▷
2. $x_r \leftarrow$ FindRegionID($x$)
3. $y_r \leftarrow$ FindRegionID($y$)
4. if rank($x_r$) > rank($y_r$)
5.   then parent($x_r$) $\leftarrow y_r$
6.   else parent($y_r$) $\leftarrow x_r$
7.        if rank($x_r$) = rank($y_r$)
8.          then rank($x_r$) $\leftarrow$ rank($x_r$) + 1
```

The merging predicate is defined as:

$$P(\mathbf{R}, \mathbf{R}') = \begin{cases} \texttt{true} & \text{iff } |\overline{\mathbf{R}'} - \overline{\mathbf{R}}| \leq \sqrt{b^2(\mathbf{R}) + b^2(\mathbf{R}')} \\ \texttt{false} & \text{otherwise} \end{cases},$$

with $b(\mathbf{R}) = 512\sqrt{\frac{1}{|\mathbf{R}|}\log\frac{|\mathbf{R}|}{3N}}$, for a 8-bit RGB color image of $N = w \times h$ pixels. Region-growing segmentation tends to yield imprecise boundaries compared to graph-cut edge-based segmentation methods [5]. Loosely speaking, as regions become bigger the mean/variance statistics reflect better the region attributes but does not tell much on whether merging two incident regions will provide a seamless merge at the region borders or not. Thus, to obtain better object boundaries from the region-merging paradigm, we rather consider the statistics of the *region crusts* (and not the full regions as in [9]). The crust of region $\mathbf{R}$ is defined as the pixels *within* distance c to its border $\partial\mathbf{R}$. (Thus for large enough c, there is no difference between regions and their crusts, and the algorithm remains the standard region-merging.) Furthermore, we only need to update the crust's mean/variance statistics when we merge regions. Updating the crust statistics is done by retrieving the pixels of the merged region belonging to its crust using a slightly modified flood-filling algorithm.[3]

Since fully automatic segmentation may give results that differ from human perception, we provide a simple user-steered mechanism to control interactively the segmentation. A user may input object (foreground)/background cues by pinpointing at a few pixel positions. Each time a user input some bias, the segmentation is recomputed in real-time (0.03s for VGA images on Intel® Pentium® IV 3.6 GHz), as we do not need to reinitialize nor sort the region pair order. We handle bias in our crust merging algorithm similarly to [9]. First, let us rename regions into metaregions, and define *pure metaregions* as the metaregions that do not contain any bias information. The other metaregions are said *biased* and contain at least one object/background pixel pinpointed by the user. When inspecting the adjacent metaregion pairs, we *never* merge metaregion pairs that contain both models. Pairs with both pure metaregions are handled as in the nonbias case. To decide whether to merge a biased metaregion with a pure metaregion, we choose among *all* metare-

[3]Flood-filling is used in painting systems to fill objects from a seed pixel position given a specified foreground color. Flood-filling-type segmentations are also called watershedding.

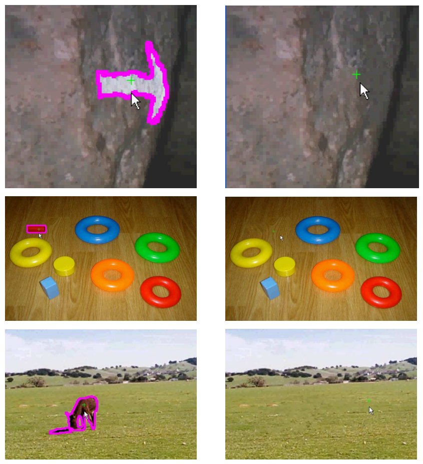
Figure 4: 1-Click removal results (fully automatic segmentation).

gions having the same bias ID (say, 1=object and 0=background), the one that statistically best matches the pure metaregion and test using the merging predicate P whether they should merge or not. At the last stage, we merge all biased object metaregions together, and all biased background metaregions altogether. Note that ClickRemoval is different from watershedding-like Photoshop's magic wand since (1) it performs global segmentation, and (2) accept both inside/outside object cues.

4. EXPERIMENTS AND CONCLUSIONS

We implemented in C++ using OpenGL® the ClickRemoval system. The full code is a mere 1000 lines, including both the novel crust segmentation algorithm and the per-pixel texture synthesis procedure [15]. Figure 4 and Figure 5 displays a few examples obtained using ClickRemoval. The accompanying video provides a sense of the UI and the system responsiveness. Although our crust-based region-growing segmentation algorithm does not provide as accurate object boundaries as edge-based graph cut methods [5], it is much faster, allows to handle *light* user-supplied bias information, and is anyway enough for our object removal application. Bias is input as a few inside/outside points (the smallest "extra" information unit) and not by strokes as in [5]. This UI is particularly advantageous for objects with many contours, since "intelligent scissors" assume in their UI that objects have a single outer contour (Figure 6). We are currently investigating further extensions to our system: pulling out object mattes from trimaps obtained by our segmentation (e.g., see [13, 6]), trading between texture synthesis and image inpainting [2], testing various texture synthesis methods (per-pixel, per-patch, per-tile, etc) while keeping the system responsiveness.

We envision such an interactive system fed by live video camera images with numerous applications in computational photography [1] and augmented/modified reality.

Acknowledgments. The Berkeley Segmentation Dataset and Benchmark (`http://www.cs.berkeley.edu/projects/vision/grouping/segbench/`). The Approximate Nearest Neighbor libary (`http://www.cs.umd.edu/~mount/ANN/`).

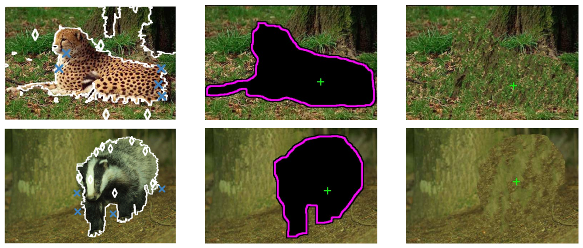
Figure 5: Result with user-input bias: a few clicks (a dozen) are enough to remove complex objects.

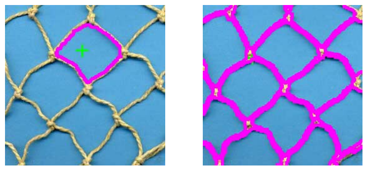
Figure 6: ClickRemoval allows to either catch a hole or remove the whole net using a single click.

5. REFERENCES

[1] W. A. Barrett and A. S. Cheney. Object-based image editing. In *SIGGRAPH*, pp. 777–784, 2002.

[2] A. Criminisi, P. Perez, and K. Toyama. Region filling and object removal by exemplar-based inpainting. *IEEE Transactions on Image Processing*, 13(9):1200–1212, 2004.

[3] A. X. Falcão, R. A. Lotufo, and G. Araújo. The Image foresting transformation. TR #IC-00-12, 2000.

[4] M. Gleicher. Image snapping. In *SIGGRAPH*, pp. 183–190, 1995.

[5] Y. Li, J. Sun, C.-K. Tang, and H.-Y. Shum. Lazy snapping. In *SIGGRAPH*, pp. 303–308, 2004.

[6] S. Lin, Q. Zhang, and J. Shi. Alpha estimation in perceptual color space. In *Proc. IEEE Acoustics, Speech, and Signal Processing (ICASSP)*, 2005.

[7] E. N. Mortensen and W. A. Barrett. Intelligent scissors for image composition. In *SIGGRAPH*, pp. 191–198, 1995.

[8] E. N. Mortensen and W. A. Barrett. Toboggan-based intelligent scissors with a four-parameter edge model. In *CVPR*, pp. 2452–2458, 1999.

[9] R. Nock and F. Nielsen. Semi-supervised statistical region refinement for color image segmentation. *Pattern Recognition*, 38(6):835-846, 2005.

[10] P. Pérez, A. Blake, and M. Gangnet. Jetstream: Probabilistic contour extraction with particles. In *ICCV*, pages 524–531, 2001.

[11] A. Rosenfeld. *Picture Processing by Computer*. 1969.

[12] C. Rother, V. Kolmogorov, and A. Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. In *SIGGRAPH*, pp. 309–314, 2004.

[13] J. Sun, J. Jia, C.-K. Tang, and H.-Y. Shum. Poisson matting. In *SIGGRAPH*, pp. 315–321, 2004.

[14] K.-H. Tan and N. Ahuja. Selecting objects with freehand sketches. In *ICCV*, pp. 337–344, 2001.

[15] L.-Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH*, pp. 479–488, 2000.