EUROGRAPHICS 2002 / I. Navazo Alvaro and Ph. Slusallek
(Guest Editors)

Short Presentations

Image Editing with Intelligent Paint

J. Reese and W. Barrett

Department of Computer Science, Brigham Young University, Provo, Utah, United States

Abstract

*Intelligent Paint provides a new tool for interactively selecting image objects or regions of interest, while simultaneously applying filters or effects directly to the selection(s). By coupling adaptive cost-ordered region growing with the high-level visual expertise of the user to indicate which objects are of interest, we are able to accurately select and edit complex, non-homogeneous objects in only a few seconds using simple mouse input gestures. Prior to selection, the image is oversegmented using a watershed (tobogganing) algorithm. Watershed catchment basins are assembled into an hierarchical image partition by grouping basins using the student's t-distribution. Grouped basins, referred to as TRAPs (**T**obogganed **R**egion **A**ccumulation **P**lateaus), reasonably capture sub-object detail. Object selection is accomplished by user-steered, adaptive, cost-ordered collection of TRAPs. Tinting or application of filters **during** object selection provides immediate visual feedback as to what is being selected as well as the suitability of the chosen digital effect. Intelligent Paint selection compares favorably with popular commercial tools in terms of efficiency, accuracy and reproducibility with the added benefit of on-the-fly filtering, while raising the granularity of image editing from the pixel to the object-level. Novel tools for object-centered image editing such as Intelligent Eraser and Intelligent Clone tools are also possible with this technique.*

Categories and Subject Descriptors (according to ACM CCS): I.3.4 [Computer Graphics]: Graphics Editors, Paint Systems I.3.6 [Computer Graphics]: Interactive Techniques I.4.3 [Image Processing and Computer Vision]: Enhancement, Filtering I.4.6 [Image Processing and Computer Vision]: Segmentation, Region growing, partitioning

1. Introduction

Current tools for image editing require the user to either 1) apply effects to the entire image, or 2) explicitly preselect the intended object or region to be modified, then (in a separate step) apply the desired operation to the selection. For example, image editing packages such as Photoshop and The GIMP allow users to select regions using one or more user-guided selection tools (rectangle/ellipse selection, Magic Wand, Lasso, Magnetic Lasso, Bezier Paths, etc.) and then apply artistic filters or digital effects to the selection.

In addition to providing a simple paint interface for object selection, Intelligent Paint (IP) provides a novel process for interactive application of special effects (recoloring, blurring, artistic filters, distortions, etc.) to interactively selected objects or regions. The selection is both implicit and simultaneous. IP is unique among selection tools in allowing the user to apply filters directly to objects of interest *while* the effect itself provides in-process visual feedback affording control of the selection as it is occurring. Thus, when the selection is complete, so is the effect (Figure 1).

Popular selection tools include manual paint brushes and lassos that are easy to implement and widely available but rely heavily on human input[1, 2]. Where object selection requires greater accuracy the user must work slowly and/or zoom to higher levels of magnification. Geometric (rectangle/ellipse) selection tools form selections using only start and end (mouse press and release) points thereby simplifying human-input. Path tools provide greater flexibility by utilizing curves to create and edit selections that are not constrained to a particular geometric shape, but exhibit some smoothness in boundary curvature. However, all curve-constrained tools rarely adhere precisely to object boundaries in real-world scenes, and still require some type of manual drawing, tracing, or placement of vertices to position boundaries. Such tools are limited by the precision with which users can place boundaries at a given zoom factor where operator bias, and variability in human input limit both accuracy and reproducibility. With manual selection tools, the user is mainly concerned with the details of object delineation rather than high-level object identification.

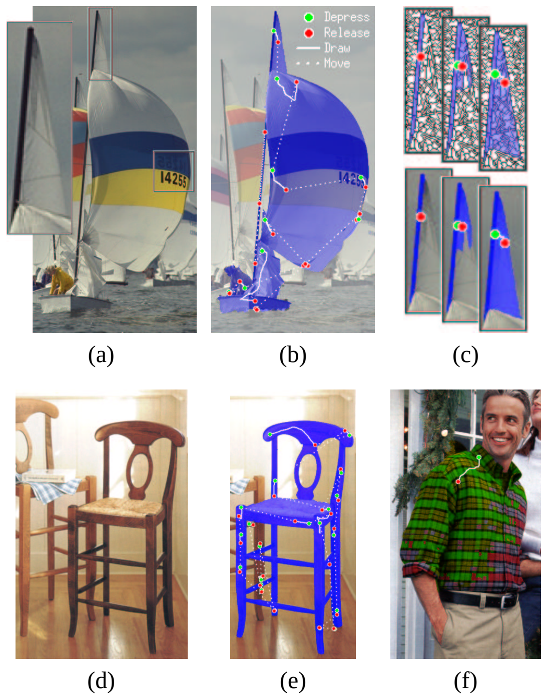

Figure 1: *Intelligent Paint connect-and-collect object selection and object-centered image editing. (a) Original boat image, with insets. Size: 490x630. (b) Sailboat selected from complex background of similar objects [18 clicks, 25 seconds] green circle→mouse-down, red circle→mouse-up, solid line→mouse drag, dashed line→mouse move. (c) Selection of part of mast and sail in progress. (d) Original chairs image. Size: 370x580. (e) Selected chair. [22 clicks, 30 seconds] (f) Image editing with Intelligent Paint (in-progress) — painting plaids (dye on-the-fly).*

Automated segmentation[3], clustering, and classification techniques such as watersheds[4, 5, 6] Laplacian zero-crossings[7], Bayesian classifiers[8], *k*-means[7] that do not allow human input have proven useful in automating certain segmentation tasks. But, such applications typically require controlled lighting, restricted image content, or other constraints in order to produce meaningful results. A general purpose selection tool must be able to process scenes of arbitrary complexity to separate foreground objects of interest from the remaining image background.

Vision assisted tools such as magic wands[1, 2], snakes[9, 10], active contours[11], and balloons[12] combine human interaction and vision algorithms by allowing interactive initialization prior to computing borders or regions. Magic wands and other region growing approaches typically allow manual specification of a sample seed or region after which the algorithm iteratively adds neighboring pixels based on homogeneity criteria[13, 14]. Snakes allow manual initialization of an approximate boundary which the algorithm later refines by converging toward detected boundaries. Local boundaries may be nudged and refined until the desired segmentation is achieved.

Most prior work in vision-assisted interactive object selection has taken the form of iterative adjustment of seed points, boundaries, thresholds, tuning parameters, or heuristics to incrementally refine a selection toward the desired result. These "initialize-then-process" schemes require the interface (user-input) to entirely precede the algorithm preventing coordination of human-computer interaction. Because all input is required prior to receiving any visual feedback, it can be difficult to determine *a priori* how much input is needed and what trouble spots may require extra input. Consequently, minimally interactive approaches can require too much input, produce unexpected results, and generally cannot be guided or steered by the operator. Hence, intelligent segmentation tools that interactively exploit high-level visual expertise but require minimal user interaction, become appealing.

Intelligent Scissors[15] is a boundary-based selection tool that allows the user to sweep the cursor around an object while a live-wire automatically snaps to, and wraps around detected object boundaries with real-time visual feedback[16]. While the *scissors* metaphor is ideal for *edge-based* object extraction, by intelligently cutting along object borders, it can require greater effort when applied to complex objects with interior holes or intricate boundaries as in Figures 1, 15, and 9).

Intelligent Paint adopts the interactive style of Intelligent Scissors in a *region-based* context using a *paint* metaphor to provide real-time interactivity and visual feedback [17]. The "intelligence" of the paint lies in its ability to adapt to regions within an object without spilling into similar, neighboring background objects or regions. This region-based approach allows the operator to perform high-level object *recognition* by dragging the cursor through the object with little attention toward interior holes and boundary complexity while the underlying segmentation algorithms handle the *delineation* of object boundaries. While boundary-based techniques must finalize the entire closed contour before any part of the region can be identified or edited, Intelligent Paint provides a feedback mechanism allowing filters and effects to be applied directly *during* the selection.

2. Intelligent Paint: a Selection Tool

Using Intelligent Paint is like pouring object-specific paint from a bucket (flood-fill) with important enhancements designed to improve accuracy and visual feedback while minimizing human input. The interface is very simple. The only input that is required is to press the mouse button (Button1) to begin painting at the cursor position, and then release the button when finished painting. While the button remains depressed, Intelligent Paint does three things: 1) it samples the image underneath the cursor (initial seed point), 2) it proactively flows the paint from this initial seed point toward object boundaries with real-time, pixel-by-pixel visual feedback, and 3) it interactively responds to mouse movement

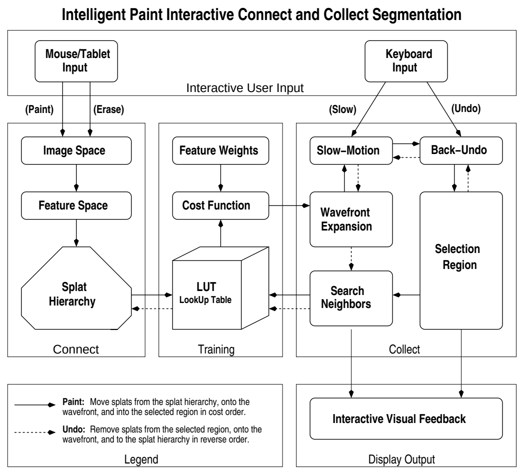
Intelligent Paint Interactive Connect and Collect Segmentation
Mouse/Tablet Input
Keyboard Input
Interactive User Input
(Paint)
(Erase)
(Slow)
(Undo)
Image Space
Feature Space
Splat Hierarchy
Connect
Feature Weights
Cost Function
LUT
LookUp Table
Training
Slow–Motion
Back–Undo
Wavefront Expansion
Selection Region
Search Neighbors
Collect
Paint: Move splats from the splat hierarchy, onto the wavefront, and into the selected region in cost order.
Undo: Remove splats from the selected region, onto the wavefront, and to the splat hierarchy in reverse order.
Legend
Interactive Visual Feedback
Display Output

Figure 2: *Intelligent Paint implementation of automated connection and interactive collection with on-the-fly training.*

by resampling the image and reflowing the paint along the entire input path treating each point along the path as a seed point. Painting terminates when the region sufficiently similar to the sample(s) has been painted, or when the mouse button is released. If the flow of paint terminates while the button remains depressed, subsequent movement without releasing the button indicates that additional painting is required and the process starts again.

We illustrate this process in the selection of the sailboat in Figure 1[a-b]. Figure 1[a-inset] shows a portion of the boat with corresponding TRAPs and three separate selections illustrated in Figure 1[c]. Note that the green circle indicates mouse-down, red circle: mouse-up, solid line: mouse drag, and dashed line: mouse move. For example, Figure 1[c-left] shows mouse-down followed by mouse-up, and the corresponding selection of the mast. Note that without mouse movement the red circle obscures the green and the flow of paint terminates automatically. Figure 1[c-middle]: object selection is extended with mouse movement. Figure 1[c-right]: additional movement captures entire object, again with automatic termination. Extending this process, with appropriate mouse drags and releases, allows the entire sailboat (and sailor) to be selected.

Intelligent Paint can be applied at the pixel level, but it is well known that pixel-based region growing algorithms tend to produce selections with small holes and irregular boundaries. Furthermore, selecting a single pixel sample/seed is not robust to variability in human input. If an outlier is chosen, the final result can be wildly unpredictable as with magic wands. Hence, we adopt a *connect* and *collect* strategy using an hierarchical segmentation algorithm to *automatically* connect (bottom-up) pixels into TRAPs, producing a fine-to-coarse hierarchy of segmentations. Meanwhile, a dynamic, cost-ordered accumulation algorithm allows the user to *interactively* collect (top-down) regions that define the object of interest. Together, these algorithms coordinate human-computer interaction, providing simultaneous object selection and object-centered image editing (Figure 2). Our approach automates tedious and error prone low-level boundary localization, while the user *always* retains interactive high-level control of object selection.

2.1. Automatically Connect

Automated segmentation techniques such as watersheds nicely partition images into relatively homogeneous regions and establish boundaries between dissimilar neighboring regions. The boundaries of objects in the original image are most likely delineated by crestlines separating the watershed catchment basins. The problem is that watersheds produce highly over-segmented partitions so that there are many unwanted crestline boundaries even within objects of interest. Many seeded, marker-based, region-merging[18], graph-based[19], scale-space[20], hierarchical[21, 5], pyramid[22] and multi-scale[23] approaches have been presented that effectively reduce unwanted boundaries and even provide hierarchies of progressively higher-level segmentations.

For our purposes in interactive object selection, we require a partitioning scheme that is 1) fast, 2) locally adaptive, 3) free of heuristic tuning parameters or thresholds, and 4) does not require *a priori* knowledge as to the number and placement of seeds or markers. It is also desirable to create an hierarchy of segmentations without generating and processing scaled copies of the original image. We produce a fine-to-coarse hierarchy of candidate segmentations largely associated with image objects at multiple resolutions (Figures 3 and 4) using our implementation of an efficient watershed (tobogganing) algorithm. Tobogganing was introduced by Fairfield[24] and extended by Yao and Hung[25]. With improvements in accuracy and efficiency, tobogganing was first used for interactive object selection in Intelligent Paint[17, 26] and subsequently applied to Intelligent Scissors[27].

2.1.1. Tobogganing

Tobogganing over-segments an image into small regions by "sliding in the derivative terrain". Given a discontinuity measure such as gradient magnitude, a slide direction (dashes in Figure 3 [top-right]) is computed for each pixel pointing toward the neighboring pixel with the lowest discontinuity. Pixels that "slide" to the same local minimum (dots) are grouped together forming a TRAP. TRAPs are similar to the catchment basins produced by watersheds, yet tobogganing is far more computationally efficient. In addition, tobogganing can be applied to a local region of interest whereas fast watershed algorithms must be applied globally, processing the entire image before meaningful results are obtained[6].

We compute discontinuity using a family of multi-scale derivative-of-Gaussian kernels to fit edges in a wide range of images. Using a two-dimensional normal distribution with a standard deviation of σ,

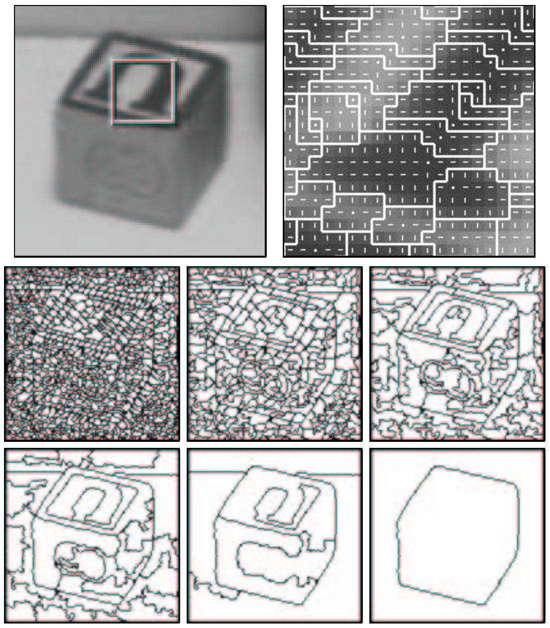

Figure 3: *Top-right: Tobogganing applied to zoomed square region of image (top-left). Dashes within each TRAP indicate gradient flow toward base point (local minimum). Below: 6 level fine-to-coarse TRAP hierarchy preserves similarity in region size at each level while retaining sub-object detail (e.g. "8", "U", block).*

$$N_S(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x^2+y^2}{\sigma^2}\right)} \tag{1}$$

the multi-scale gradient magnitude is given by

$$G(x,y) = \sqrt{max(G_\sigma(x,y))} \tag{2}$$

where

$$G_\sigma = \sum_b \left\{ \left[I_b * \frac{\partial N_\sigma}{\partial x} \right]^2 + \left[I_b * \frac{\partial N_\sigma}{\partial y} \right]^2 \right\} \tag{3}$$

is the squared gradient magnitude summed over each color band I_b of the image I, computed by convolving each I_b with a first-derivative-of-Gaussian kernel of scale σ. We interpret the discontinuity measure, $D(p,q)$ between two neighboring pixels, p and q, as the gradient magnitude at q,

$$D(p,q) = G(q) \tag{4}$$

and the minimum discontinuity neighbor of p is

$$M(p) = \min_{q \epsilon N(p)} (D(p,q)) \tag{5}$$

where $N(p)$ defines the 4-connected neighborhood of p (including p itself).

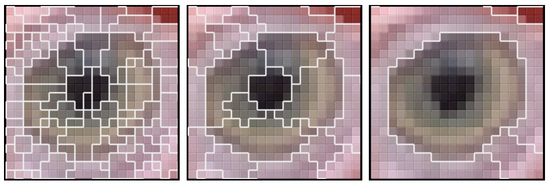

Figure 4: *"Bird's eye" view of TRAP hierarchy from (Figure 15). Left: initial TRAPs. Middle: second level TRAPs obtained by hierarchical tobogganing. Right: Third level TRAPs. Note preservation of sub-object detail (pupil, level 2; eye, level 3).*

Previous tobogganing[24, 25] and watershed[6] methods use an 8-connected neighborhood but fail to account for the increased Euclidean distance to the diagonal neighbors. This leads to spatially inaccurate region boundaries since pixels on or near a gradient ridge may "tunnel" diagonally through the ridge, where pixel corners touch, and slide down the wrong side. While a 4-connected neighborhood produces more and smaller regions, it isolates fine detail better and is about twice as fast. In our implementation, smaller regions are merged into larger regions using hierarchical tobogganing Section 2.1.2.

Given a discontinuity measure and a neighborhood, tobogganing partitions the image by assigning a unique label to each of the tobogganed regions as described in Algorithm 1.

Algorithm 1: 4-Connected Tobogganing

```
Input:
  I(p)      Image to be partitioned by tobogganing

Output:
  L(p)      Labeled TRAPs (initialized to 0 for all p)

Methods:
  D(p,q)    Discontinuity measure between p and q
  M(p)      Minimum cost neighbor of p

Algorithm:

  count := 0;             Init number of regions to 0
  label := 0;             Init current region label to 0

  for each p {            Scan pixels p in row-major order
    if (L(p)) continue;   Skip any previously labeled p
    label:=count+1;       Anticipate new local min label
    q:=p;                 Set q to begin sliding at p
    while (L(q)==0) {     Slide until q is labeled or a min
       L(q):=label;       Assign new label anticipating min
       push q;            Store q in case we need to relabel
       q:=M(q);           Slide toward lowest cost neighbor
    }                     Done sliding
    if (L(q)==label) {    If q is a min the path is labeled
       empty stack;       Clear stack for next slide-path
       count:=count+1;    Increment the number of regions
    else {                If q is labeled, relabel slide-path
       label:=L(q);       Set the label to match q
       while (stack) {    Relabel all q on the stack
         pop q;           Get next q along slide-path
         L(q):=label;     Relabel with encountered label
       }                  Done relabeling
    }                     Done labeling
  }                       Done tobogganing
```

The image is scanned in row-major order. When an unlabeled pixel is found, its neighborhood is searched to identify and slide toward the minimum discontinuity neighbor. Sliding continues until the slide-path encounters a previously labeled pixel or a local minimum. If a labeled pixel is encountered, sliding stops because the monotonically decreasing path toward the minimum has already been labeled. Hence, the encountered label is assigned to each pixel along the slide-path. If instead, the slide-path reaches an unlabeled local minimum, a unique label is assigned for this minimum and those pixels along the slide-path leading to it. We have modified the original tobogganing algorithm[24, 25] to begin labeling each slide-path with a new label anticipating a new local minimum so that we only relabel the slide-path if an old label is encountered. This results in a speed-up of approximately 20 percent since the first slide-path encountered for a new region is traversed only once.

2.1.2. Hierarchical Tobogganing

Tobogganing at pixel resolution creates highly over-segmented regions. However, by applying the tobogganing algorithm to TRAPs rather than pixels, we can produce a fine-to-coarse hierarchy of segmentations. Since we combine existing TRAPs rather than recomputing new regions with a looser grouping criterion or threshold, boundaries in subsequent levels correspond exactly to a subset of the prior boundaries (Figure 3).

Hierarchical tobogganing proceeds in the same fashion as pixel-based tobogganing with the exception that we are grouping TRAPs rather than pixels. We simply redefine the neighborhood and the discontinuity measure for Algorithm 1. We define the neighborhood $N(p)$ of TRAP p as the set of all TRAPs $q \neq p$ such that q is 4-connected to p. The discontinuity measure $D(p,q)$ between TRAPs p and q is computed using the student's t-distribution[28] which estimates the probability that two measured distributions have different means. As the probability of different means between TRAPs p and q increases, so does the discontinuity between TRAPs p and q. $D(p,q)$ is computed using the student's t-distribution as

$$D(p,q) = p(t|\nu) = \frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})\sqrt{\pi\nu}} \int_t^{\infty} \left(1+\frac{t^2}{\nu}\right)^{-\left(\frac{\nu+1}{2}\right)} \tag{6}$$

where t is the student's t-score and ν is the degrees of freedom given by

$$t = \frac{\overline{x}_p - \overline{x}_q}{\sqrt{\sigma_p^2/N_p + \sigma_q^2/N_q}} \tag{7}$$

and

$$\nu = \frac{[\frac{\sigma_p^2}{N_p} + \frac{\sigma_q^2}{N_q}]^2}{\frac{[\sigma_p^2/N_p]^2}{N_p-1} + \frac{[\sigma_q^2/N_q]^2}{N_q-1}} \tag{8}$$

respectively. For TRAPs p and q, variables $\overline{x}$, σ^2, and N are the mean, variance, and number of samples of each TRAP. In our implementation, we automatically compute the first six levels of the hierarchy and allow the user to select the appropriate level of object detail (level two is the default).

2.2. Interactively Collect

Once the appropriate level of detail has been selected, the task is to collect, in cost order, with minimal user input, the TRAPs corresponding to the object of interest. Cost-ordered TRAP collection begins (Algorithm 2) when the operator points, clicks and drags the cursor within the object of interest as illustrated in Figure 1[c]. All sampled TRAP(s) touched along the cursor path are placed at the head of a cost-ordered list referred to as the wavefront, W, to seed the graph expansion.

Cost-ordered TRAP collection is begun by initializing a 3D histogram or look-up-table (LUT) with the RGB values of the pixels within the TRAP first pointed to. The LUT is immediately augmented by pixels from TRAPs collected in the expansion, and subsequently updated with pixels from TRAPs touched by the drag. Adjacent traps are added in cost order, where the (local) cost of a TRAP is determined based on the similarity of its RGB values to those found in the LUT. In our current implementation the local cost $L(p)$ is computed as

$$L(p) = \frac{1}{N} \sum_{x \in T} \frac{1}{LUT(x_r, x_g, x_b)} \tag{9}$$

where the RGB value (x_r, x_g, x_b) of each of the N pixels x in TRAP T indexes the LUT. Thus, if T's histogram resembles the current LUT, the cost $L(p)$ of adding T will be comparatively small.

In our implementation, the LUT is a 3D (64x64x64) histogram (initialized to 1). We implement W as a circular, double-linked list with one element (bin) for each possible discrete local cost $L(p)$. Interactively sampled TRAPs are assigned zero cost and inserted into the bin at the *head* of the sorted wavefront. Local costs $L(p)$ are computed for neighboring TRAPs $N(p)$ not yet collected or on the wavefront. Neighbors are then added into the bin at position $head + L(p)$.

Once initialized, the expansion continues by removing lowest cost TRAPs from the head of the list, marking them as collected, and adding any new neighboring TRAPs as before. When the bin at the head of the list is empty, the head is advanced to the next non-empty bin until W is empty or the mouse button is released. As the *head* advances through the circularly-linked list, it is always positioned at the current cumulative cost so that adding new entries at $head + L(p)$ maintains the cumulative-cost ordering of the list. The local cost $L(p)$ is computed using a dynamically updated cost look-up-table (Section 2.3).

Algorithm 2: Cost-ordered TRAP Collection

```
Input:
  s          Seed or sample TRAP
Output:
  L(p)       Label TRAP p (BG, FG, Wavefront, Cooled)
Methods:
  W          Wavefront cumulative cost-ordered list of TRAPs
  N(p)       Neighbors of TRAP p
  L(p)       Local cost to add TRAP p to wavefront
  C(p)       Cumulative cost to add TRAP p to wavefront
  M(T_i,T_g) Min(intensity T_i, gradient T_g) cooling thresholds

Algorithm: Cost-ordered TRAP Collection
  C(s):=0;                      Init cumulative cost of s to 0
  L:=BG;                        Init output labels to BG
  W:=s;                         Place TRAP s on the Wavefront
  while W {                     While TRAPs are on Wavefront
    p:=min(W);                  Remove min cost TRAP from W
    L(p):=FG;                   Mark TRAP p as FG (collected)
    for q in N(p) {             For each neighbor q of p
       if L(q)==BG {            If q has not yet been processed
         if L(q)>M(T_i,T_g)     Test if q should be cooled
           L(q):=Cooled;        If so, then mark q as Cooled
         else {                 Compute cost and add q to W
           C(q):=C(p)+L(q);     Assign cumulative cost to q
           L(q):=Wavefront      Insert q to W in sorted order
         }                      Done adding q to Wavefront
       }                        Done processing neighbor q
    }                           Done expanding neighbors
  }                             Done collecting TRAPs
```

Steerable, interactive region accumulation with cost-ordered TRAP collection and in-process visual feedback allows the operator to guide the object recognition process. A *cumulative* cost ordering C provides a sense of local control in proximity to the user-positioned cursor not found in most region growing techniques.

Our cumulative, cost-ordered region expansion is adapted from the unrestricted graph expansion used by the original Intelligent Scissors[15, 16] with a few important distinctions. First, we raise the granularity and operate within the TRAP-based framework rather than at pixel-level[17]. Second, we use the graph expansion to collect TRAPs within the desired object rather than to select boundary segments from a collection of piecewise optimal paths[27]. Third, the weighted graph is formulated by treating TRAPs as weighted nodes with edges connecting neighboring TRAPs. Fourth, we use local region-based costs. Fifth, we do not need to expand the graph throughout the entire image, thus TRAP collection terminates either interactively by releasing the mouse button or automatically via adaptive cooling (Section 2.3). And sixth, the user is provided real-time visual feedback and steers the graph expansion process itself rather than only interacting with pre-computed results.

2.3. On-the-fly training

On-the-fly training employs several methods that enable Intelligent Paint to dynamically adapt to non-homogeneous regions of interest, thereby improving overall efficiency, accuracy, and robustness in obtaining the desired selection. These methods include the use of a dynamically updated cost look-up-table, adaptive cooling, interactively spawned seed or sample TRAPs, and automatic cursor snapping.

1. ***Dynamically Updated Cost Look-Up-Table (LUT)*** allows local trap costs to vary throughout the process based on the current statistical composition of the expanding region using what is learned at early stages of the segmentation to guide subsequent processing.
2. ***Adaptive Cooling*** freezes, or curtails growth at given positions along the wavefront by removing from the wavefront, those TRAPs whose costs (intensity or gradient) are too high relative to the sampled and collected region.
3. ***Dynamically Spawned Seed TRAPs*** allows paint to immediately and automatically flow from every TRAP touched by human input as the user interactively paints the object of interest.
4. ***Automatic Cursor Snapping*** automatically warps (translates) the cursor small distances to assist in painting fine-detail regions that are not yet processed.

3. Intelligent Paint: for Image Editing

For the most part, editing image objects with Intelligent Paint by applying filters and effects directly into intelligently selected regions is straightforward. The user simply chooses the desired effect and begins painting. Intelligent Paint expands the effect until it reaches object boundaries giving the visual impression of intelligently pouring the effect into the desired region. This novel aspect of Intelligent Paint also provides immediate visual feedback as to the appropriateness of the filter used.

A complete discussion of implementation details for the many filters and effects possible with Intelligent Paint is not included here, but we note several important observations. First, applying point effects such as brightness, contrast, hue, and saturation adjustments is straightforward. Instead of tinting or flood-filling the region during selection, the effect is simply drawn into the image instead, and the visual feedback is *in* the effect. Second, neighborhood operators such as blur or sharpening (convolutions in general) require special processing along selection boundaries within half of the width of the neighborhood size so that background regions are not contaminated in the processing. Third, the user may choose *constrained* or *cumulative* application of effects to either 1) limit processing of each pixel to one application of the effect (one coat of paint), or 2) apply the effect repeatedly each time the pixel is selected (multiple coats of paint). Finally, multiple filters and effects can be applied simultaneously by cascading or stacking filters to combine them into a single operation.

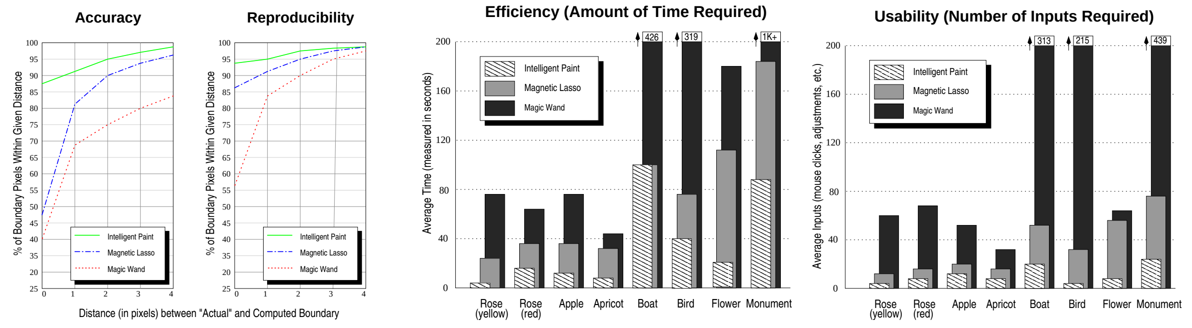

Figure 5: *Summary of user-study results with 6 participants, and 504 object selection tasks (Figure 15). Accuracy (spatially correct placement) and Reproducibility (inter-user and intra-user consistency)[left]. Efficiency (amount of time)[middle]. Usability (number of inputs; mouse clicks, undos, and parameter adjustments)[right].*

4. Results and Discussion

The basic operation of Intelligent Paint is to identify and select image objects by pointing at, and dragging the cursor through the object with little or no user effort required to localize object boundaries (Figures 6 - 9). The simplest effect is to repaint a previously edited region with original image data thereby erasing any previously applied effect(s). This provides a powerful Intelligent Eraser tool that can be used to remove unwanted effects, or spills that may have crossed through object boundaries. The use of Intelligent Paint to simulate depth-of-field by selectively applying a Gaussian blur to the background (crowd) is illustrated in Figure 10. Figure 11 demonstrates the direct application of a hue shift painted onto the body of the colorful parrot (left), a brick texture poured onto the background ("behind") the bird (middle), and a more convincing "stacked-filter" that simultaneously paints and desaturates the brick wall for greater realism. These effects are accomplished with 3 to 8 clicks in 5 to 10 seconds. Application of a variety of well known digital filters is demonstrated in Figure 12. An Intelligent Clone tool is illustrated in Figure 13 where the user simply clicks a target location (red "x"), then starts painting from the source region (green "x"). The selection obtained from the source is auto-magically painted into the offset (translated) coordinates at the destination. Finally, if cloning is not controversial enough and one wants to really make waves, artistic distortions such as ripples or waves may be painted directly into otherwise peaceful bodies of water Figure 14.

In a study consisting of a variety of selection tasks, Intelligent Paint was compared (as a selection tool) with the popular magic wand and magnetic lasso tools in Adobe Photoshop. These tools were evaluated in terms of accuracy, reproducibility, efficiency, and usability (Figure 5). The selection tasks were performed in random order by 6 participants accustomed to using both magic wand and magnetic lasso.

504 object selection tasks (84 tasks for each of the 6 participants) were used in the study. Participants were required to extract 12 objects of interest (Figure 15) from synthetic and real-world images, with each of the 3 selection tools, 3 times with each tool.

The synthetic image (Part 1) contains shapes with varying degrees of boundary curvature and region complexity. The original binary synthetic image is processed (blur and noise added) to simulate the blurring and noise of image capture hardware and to test the robustness of Intelligent Paint. Two processed images computed from the binary original with increasing levels of noise and blur are tested. The still-life (Part 2) image contains objects that may be selected fairly quickly. Other real-world images (Part 3) contain objects of varying complexity and background which were segmented in a few seconds to a few 10's of seconds with only a few mouse clicks. Furthermore, for an experienced user these times decrease by a factor of two.

Hierarchical Tobogganing is well suited to Intelligent Paint because it produces segmentations that do not depend on heuristic parameters or thresholds, it is sensitive to local features in the image data, it is computationally efficient, it is terminally order independent so that it may be parallelized or computed on-the-fly as needed for local regions, and it tends to produce similarly sized regions at each level of the hierarchy (Figure 4). Incrementally raising the granularity of the basic unit of operation above the pixel level provides a richer feature set to more accurately delineate boundaries at subsequent higher-level segmentations.

The problem with automated hierarchical techniques (including this one) is that as the granularity increases, meaningful boundaries are eventually lost. Intelligent Paint allows the operator to interactively specify a granularity within the hierarchy (using the *up* and *down* arrow keys) based on the size and detail of objects to be selected.

5. Conclusion and Future Work

We have introduced a tool that performs object selection and editing by simply dragging the cursor through the region of interest. High accuracy in segmentation is achieved by automatically establishing a fine-to-coarse hierarchy of spatially accurate boundaries between regions. High reproducibility is obtained by allowing the user to direct the segmentation while the algorithm snaps to, and localizes the object boundary. Selections require less time and fewer inputs as the process proceeds automatically where no guidance is needed, but responds immediately to input when steering is provided. In addition to simplifying the process of object selection, Intelligent Paint provides a common framework for Intelligent Eraser and Intelligent Clone (rubber-stamp) tools. Finally, Intelligent Paint provides a novel tool for object selection with simultaneous application of digital effects.

Object selection through collection of TRAPs in cumulative cost order avoids the pitfalls, including holes and ragged boundaries, of strictly local connected component region growing schemes, while providing a type of globally optimal robustness in object definition. Unlike many semi-automated approaches to segmentation, Intelligent Paint uses a "directive" rather than a "corrective" metaphor, tightly interleaving, and judiciously balancing user guidance and algorithmic boundary localization. Thus, rather than engaging the user in repetitive compute-object, correct-object cycles, Intelligent Paint shows great respect for all user input, supporting economy of effort, so that object selection almost always occurs in one pass.

Work is ongoing to explore simultaneous interaction at multiple levels within the TRAP hierarchy to capture fine detail on the perimeter of the expanding wavefront while still operating on higher-level components within the object. Selection boundaries could be processed using an edge-model[27] to compute sub-pixel accuracy and alpha channel. Modes of interaction for extending this technique to video sequences and volume data are being considered.

References

1. Adobe Systems Incorporated. *Adobe Photoshop Version 6.0 User Guide (Chapter 5: Selecting)*, 2000. 1, 2
2. Carey Bunks. *Grokking the GIMP*. New Riders, 2000. 1, 2
3. R.M. Haralick and L.G. Shapiro. SURVEY: Image Segmentation Techniques. *Computer Vision, Graphics, and Image Processing*, 29(1):100–32, 1985. 2
4. S. Beucher and C. Lantejoul. Use of Watersheds in Contour Detection. In *International Workshop on Image Processing, Real-Time Edge and Motion Detection/Estimation, CCETT/INSA/IRISA, IRISA Report*, volume 132, pages 2.1–2.12, 1979. 2
5. L. Najman and M. Schmitt. Geodesic Saliency of Watershed Contours and Hierarchical Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(12):1163–73, December 1996. 2, 3
6. L. Vincent and P. Soille. Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):583–598, June 1991. 2, 3, 4
7. M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis, and Machine Vision)*. ITP – International Thomson Publishing Inc., 1998. 2
8. S. Shah and J.K. Aggarwal. A Bayesian Segmentation Framework for Textured Visual Images. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1014–20, June 1997. 2
9. M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active Contour Models. In *Int. Conf. on Computer Vision*, pages 259–68, June 1987. 2
10. S.C. Zhu and A. Yuille. Region Competition: Unifying Snakes, Region Growing, and Bayes/MDL for Multiband Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(9):884–900, September 1996. 2
11. V. Caselles, R. Kimmel, and G. Sapiro. Geodesic active contours. In *International Conference on Computer Vision (ICCV '95)*, pages 694–699, 1995. 2
12. L.D. Cohen. On Active Contour Models and Balloons. In *Computer Vision, Graphics, and Image Processing. Image Understanding*, volume 53, pages 211–218, 1991. 2
13. R. Adams and L. Bischof. Seeded Region Growing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6):641–7, June 1994. 2
14. T. Pavlidis and Y.T. Liow. Integrating Region Growing and Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(3):225–33, March 1990. 2
15. E.N. Mortensen and W.A. Barrett. Intelligent Scissors for Image Composition. In *ACM SIGGRAPH Conference on Computer Graphics*, pages 191–8. Addison Wesley Longman, August 1995. 2, 6
16. E.N. Mortensen and W.A. Barrett. Interactive Segmentation with Intelligent Scissors. *Graphical Models and Image Processing*, 60(5):349–384, September 1998. 2, 6
17. L.J. Reese. *Intelligent Paint: Region-Based Interactive Image Segmentation*. Masters Thesis, Brigham Young University, Dept. of Computer Science, 1999. 2, 3, 6
18. K. Haris, S.N. Efstratiadis, and Maglaveras N. Watershed-based image segmentation with fast region merging. In *IEEE Int. Conf. on Image Processing*, volume 3, pages 338–342, October 1998. 3
19. K. Saarinen. Color Image Segmentation by a Watershed Algorithm and Region Adjacency Graph Processing. In *IEEE Int. Conf. on Image Processing*, volume 3, pages 1021–25, November 1994. 3
20. P.T. Jackway. Gradient Watersheds in Morphological Scale-Space. *IEEE Transactions on Image Processing*, 5(6):913–921, 1996. 3
21. J.M. Gauch. Image segmentation and analysis via multiscale gradient watershed hierarchies. *IEEE Transactions on Image Processing*, 8(1):69–79, 1999. 3
22. A.S. Wright and S.T. Acton. Watershed Pyramids for Edge Detection. In *IEEE Int. Conf. on Image Processing*, volume 2, pages 578–81, 1997. 3

23. O.F. Olsen and M. Nielsen. Multi-scale Gradient Magnitude Watershed Segmentation. In *Int. Conf. on Image Analysis and Processing*, volume 1310, pages 6–13, September 1997. 3

24. J. Fairfield. Toboggan Contrast Enhancement for Contrast Segmentation. In *Int. Conf. on Pattern Recognition*, pages 712–16, June 1990. 3, 4, 5

25. X Yao and Y.P. Hung. Fast Image Segmentation by Sliding in the Derivative Terrain. In *Proceedings of the SPIE*, volume 1607, pages 369–79, November 1992. 3, 4, 5

26. E.N. Mortensen and L.J. Reese. Intelligent Selection Tools. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 776–777, June 2000. 3

27. E.N. Mortensen and W.A. Barrett. Toboggan-Based Intelligent Scissors with a Four-Parameter Edge Model. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 452–458, June 1999. 3, 6, 8

28. W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 2 edition, 1982. 5

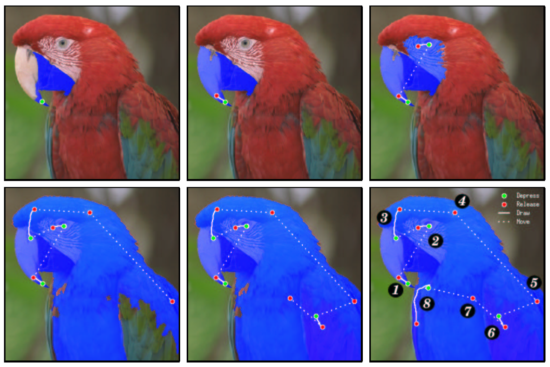

Figure 6: *Sequence of images (row-major) showing input (green circle:mouse-press, red circle:mouse-release, solid line:mouse-drag, dashed line:mouse-move) to select object (or sub-objects) in upto 8 clicks with little mouse-dragging (interactive sampling). Short strokes sample fewer TRAPs into the LUT resulting in more controlled painting.*

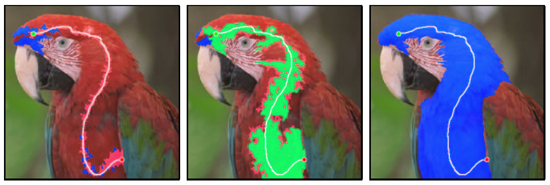

Figure 7: *Continuous mouse drag samples many TRAPs into the LUT resulting in broader painting. On mouse-press: green circle (left), paint immediately starts to flow. TRAPs touched by brush stroke are immediately added to the expanding wavefront tinted red (left). Expanding region tinted green continues to grow (middle). Selected region tinted blue when finished (right).*

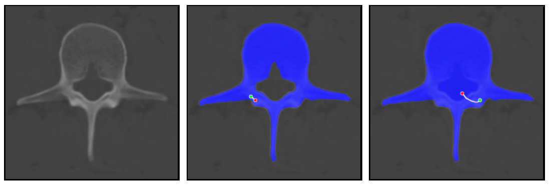

Figure 8: *Lumbar spine original (left), selected preserving hole [1 click, < 1 second] (middle), and filling hole [1 click, < 1 second] with a little more sampling (right).*

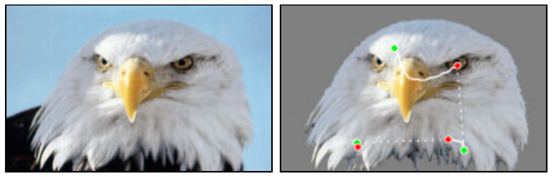

Figure 9: *Selection of eagle head; trivial with Intelligent Paint, tedious with Magnetic Lasso, exhausting with Magic Wand. Note that we do not trivially select the background then invert.*

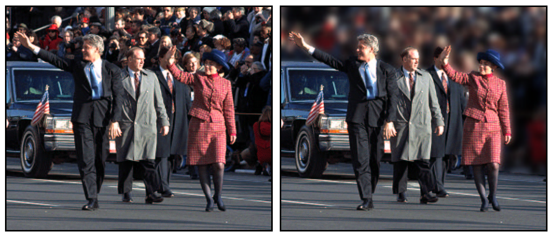

Figure 10: *Simulating depth-of-field by painting with blur effect. Background of original image (left) blurred by simultaneous selection/painting with Gaussian filter (right).*

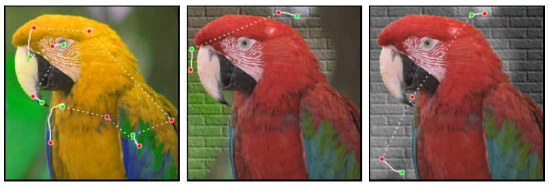

Figure 11: *Recoloring using hue-shift effect painted onto parrot (left). Brick texture being painted onto background (middle). More believable background with brick texture simultaneously painted and desaturated with stacked filters (right). See Figure 15 for original bird image.*

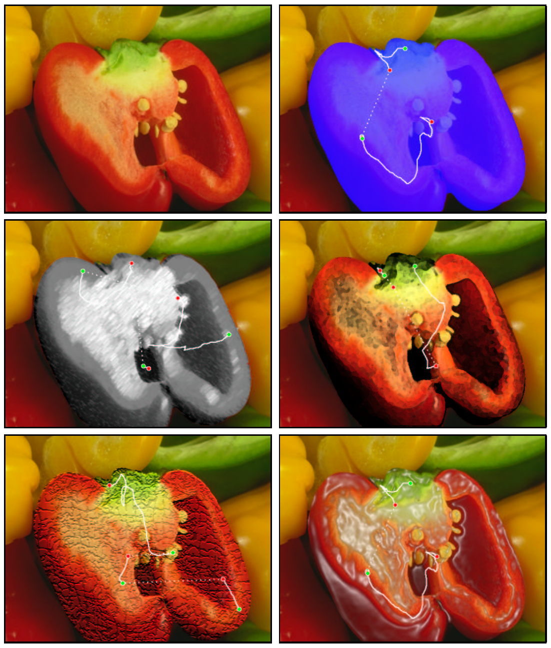

Figure 12: *In row-major order: original image [Size: 540x420], tinted selection, chalk, fresco, craquelure, and plastic wrap effects painted directly onto the pepper without separate selection. [2 to 3 clicks, 3 to 5 seconds]*

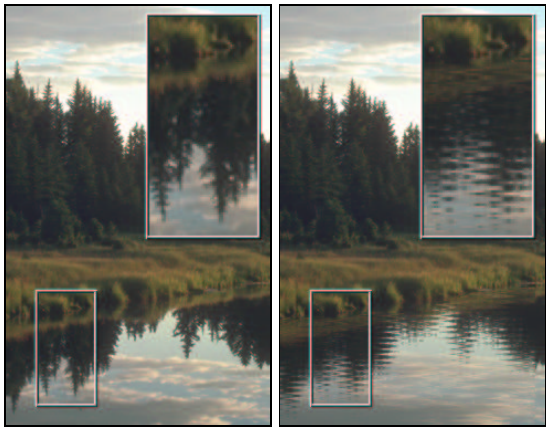

Figure 14: *Artistic distortion (ripple or wave filter) painted directly onto selection.*

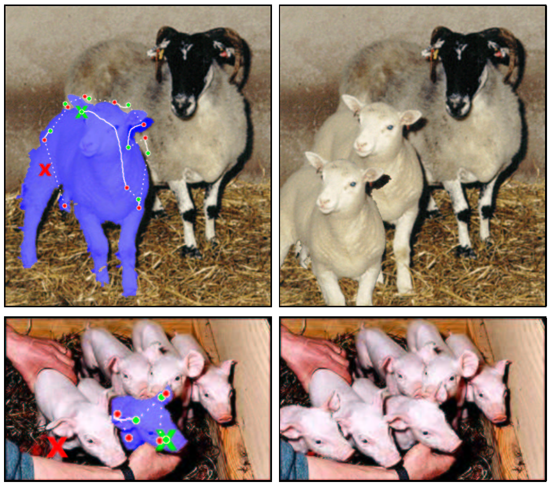

Figure 13: *CTRL-click at red "x" to mark target destination: then intelligently clone from source starting at green "x". Cloning Dolly's clone (top) [9 clicks, 20 seconds], 3 little-pigs, or 4, or 5, or 6, ... (bottom) [5 clicks, 10 seconds] (bottom-right).*

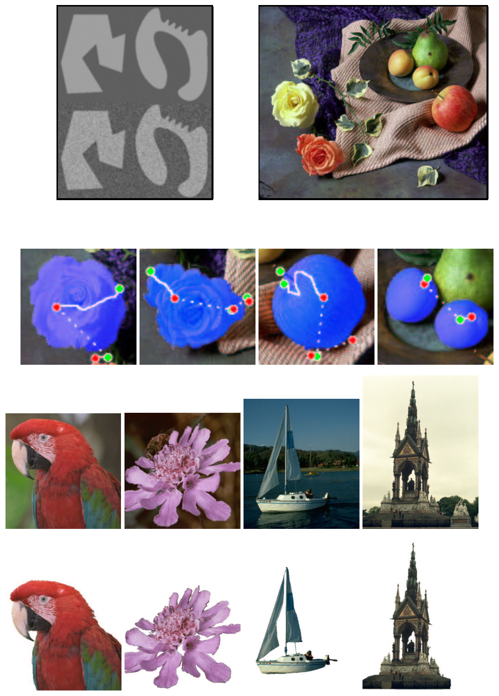

Figure 15: *Images from user study Part 1 (top-left) and Part 2 (top-right) with samples of user-input and extracted objects from Part 2 (middle), and images from Part 3 (bottom).*