

TECHNICAL REPORT

Towards Paint and Click: Unified Interactions for Image Boundaries

Brian Summa, Amy A. Gooch, Giorgio Scorzelli, and Valerio Pascucci

UUSCI-2014-004

Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, UT 84112 USA
December 15, 2014

Abstract:

Image boundaries are a fundamental component of many interactive digital photography techniques, enabling applications such as segmentation, panoramas, and seamless image composition. Interactions for image boundaries often rely on two complimentary but separate approaches: editing via painting or clicking constraints. In this work, we provide a novel, unified approach for interactive editing of pairwise image boundaries that combines the ease of painting with the direct control of constraints. Rather than a sequential coupling, this new formulation allows full use of both interactions simultaneously, giving users unprecedented flexibility for fast boundary editing. To enable this new approach, we provide technical advancements. In particular, we detail a reformulation of image boundaries as a problem of finding cycles, expanding and correcting limitations of the previous work. Our new formulation provides boundary solutions for painted regions with performance on par with state-of-the-art specialized, paint-only techniques. In addition, we provide instantaneous exploration of the boundary solution space with user constraints. Furthermore, we show how to increase performance and decrease memory consumption through novel strategies and/or optional approximations. Finally, we provide examples of common graphics applications impacted by our new approach.

Towards Paint and Click: Unified Interactions for Image Boundaries

B. Summar and A. A. Gooch and G. Scorzelli and V. Pascucci

Scientific Computing and Imaging Institute & University of Utah

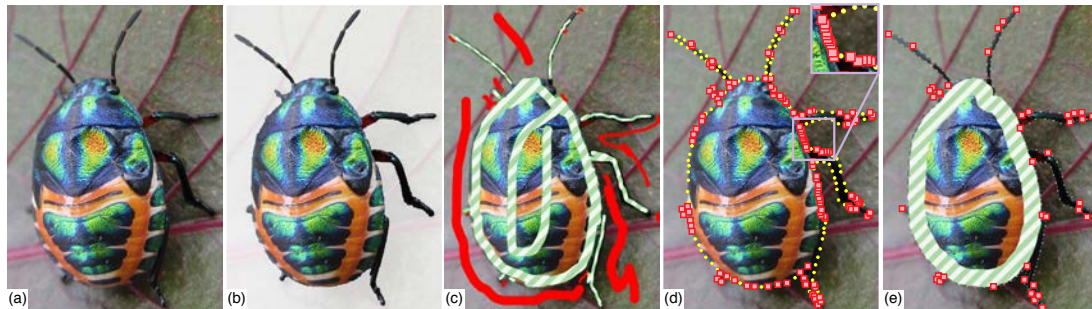


Figure 1: We provide a novel unified interaction for pairwise image boundaries that combines both paint and constraint-based user edits. (a) Input image. (b) Desired segmentation. (c) Using only painting edits, a user can use include (green striped) and exclude (red) annotations to select the object. These annotations can be numerous and tedious for fine features such as the legs and antennae. (d) Using only constraints (or anchors), a user can click (red) control points to form the object's boundary. Even with automatic constraints (yellow), many clicks are required. (e) Our unified approach allows users to mix the complimentary editing metaphors leading to a more flexible and faster experience.

Abstract

Image boundaries are a fundamental component of many interactive digital photography techniques, enabling applications such as segmentation, panoramas, and seamless image composition. Interactions for image boundaries often rely on two complimentary but separate approaches: editing via painting or clicking constraints. In this work, we provide a novel, unified approach for interactive editing of pairwise image boundaries that combines the ease of painting with the direct control of constraints. Rather than a sequential coupling, this new formulation allows full use of both interactions simultaneously, giving users unprecedented flexibility for fast boundary editing. To enable this new approach, we provide technical advancements. In particular, we detail a reformulation of image boundaries as a problem of finding cycles, expanding and correcting limitations of the previous work. Our new formulation provides boundary solutions for painted regions with performance on par with state-of-the-art specialized, paint-only techniques. In addition, we provide instantaneous exploration of the boundary solution space with user constraints. Furthermore, we show how to increase performance and decrease memory consumption through novel strategies and/or optional approximations. Finally, we provide examples of common graphics applications impacted by our new approach.

Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques

1. Introduction

We provide a novel, unified approach for interactively editing pairwise image boundaries, combining the ease of painting interactions with the direct control of constraints. Boundaries that define where one image region ends and an-

other begins are crucial in image composition and editing applications. These boundaries are often automatically or semi-automatically computed to minimize or maximize the transition between regions, removing the need to meticulously edit boundaries pixel by pixel. In particular, our

work targets user interaction for the automatic and semi-automatic construction of *pairwise boundaries* (boundaries between two images or an image with itself), which have been used with great success for a variety of research areas such as texture synthesis [EF01, CSHD03], digital panoramas [Dav98, LZW04, STP12], seamless pasting [JSTS06], and image segmentation [MB95, MB98, LSTS04].

User interaction for image boundaries often takes one of two forms, defined by the underlying core algorithm. First, *minimum cut* approaches allow users to edit boundaries by painting and, second, *minimum path* approaches allow users to edit boundaries by the creation and manipulation of constraints. Figure 1c illustrates Adobe Photoshop's quick selection painting interactions being used to select the beetle of Figure 1a; similar interactions are often used for minimum cut implementations. Selecting the beetle requires over 30 interactions, which are most challenging in areas with fine features, such as the antennae and legs. These features require a user to carefully trace their interior and/or exterior, which can be tedious and error-prone. Figure 1d illustrates adding constraints via Photoshop's magnetic lasso tool, an approach similar to most minimum path implementations. Red constraints are user clicks and yellow constraints are automatically added by the software. For areas such as the beetle's body in Figure 1d, the amount of points needed to be added by the user can be excessive and the interaction process slow. As the example in Figure 1 illustrates, the benefits and drawbacks of the two approaches are often complementary. For example, the antennae that need tedious painting require only a small number of user constraints and the laborious constraints on the beetle's body are easily avoided with a simple painting annotation.

The complementary nature of painting and constraints speaks to the need for a unified approach to image boundaries. For ease of implementation, most methods support painting and constraint schemes independently. After computing a boundary for a given interaction scheme (painting or constraints), the boundary is finalized and the paintings or constraints are discarded. Such a step-by-step pipeline leads to new edits overruling previous edits. As shown in our companion video, successive painting may remove a well-placed boundary formed by constraints or a previous painting. Painting and constraints, working together in a single approach, lead to fewer interactions overall. For example, an initial coarse painting of the beetle's body reduces the constraints needed to select its legs to just a few clicks, since the operation only moves an already semi-optimal boundary.

Our unified technique provides users with the ability to mix interactions and choose the interaction that best fits their current task without loss of previous edits. As supplemental material, we provide video captures of our new approach, which reduces the time to select the beetle in Figure 1 by almost half when comparing our unoptimized research code to pure constraints and painting with Photoshop. Even when

compared to combining both interactions independently, our new approach improves editing time significantly.

In this paper, we will detail our new, unified approach for image boundaries, describe how to compute boundaries efficiently, and provide example image processing applications. Several technical innovations were necessary to achieve our approach. In particular, the contributions of this paper are:

- A robust formulation of the optimal boundary problem guaranteed to find the minimum boundary;
- A fast and parallel algorithm to find the minimum boundary;
- A novel extension of our boundary mechanics allowing users to add constraints to the minimum boundary with instant feedback;
- A strategy to combine independent minimum boundaries that allows users intuitive editing with multiple painting annotations;
- Novel acceleration and memory reduction strategies;
- Practical applications for our new unified approach.

2. Related Work

Given its fundamental use in image processing, the automatic computation of image boundaries has an extensive body of work. Below, we concentrate on how interaction has been used to aid automatic solutions or provide boundary solutions semi-automatically.

Painting Interaction. Minimum cut algorithms, such as Graph Cuts [BVZ01, BK04, KZ04], compute boundaries via an optimization often with a user's initial painted annotations as input. A painting interaction [RKB04, LSTS04, LSGX05, JSTS06, NFK07, VN08, LSS09, LS10, TGV13] has the benefit of a metaphor (*include* and *exclude* painting) that is easily understood by most users. Paint-based interactions provide a user with quick manipulations, but are often only a front-end to an expensive, iterative optimization. Additional annotations and edits result in a full recomputation of the solution, which can be time consuming over many edits. Even if a solution can be produced quickly, more annotations may be required to resolve ambiguous boundaries [LSTS04], an ill-defined energy specification, or places where multiple aesthetically valid solutions are possible [STP12]. The previous work of Li et al. [LSTS04] has shown the need for constraints in painting-based approaches for resolving boundary ambiguities. If the desired features are fine, the additional careful annotation can be tedious and require almost the same effort as the manual editing of the image masks. Our work provides a fast and robust algorithm for painting interactions with constraints.

Constraint Interaction. Interaction using minimum paths and constraints, first seen in Mortensen and Barrett [MB95, MB98], involves the computation of minimum path trees from user-defined points (constraints). Adding new constraints simply requires a traversal of the precomputed

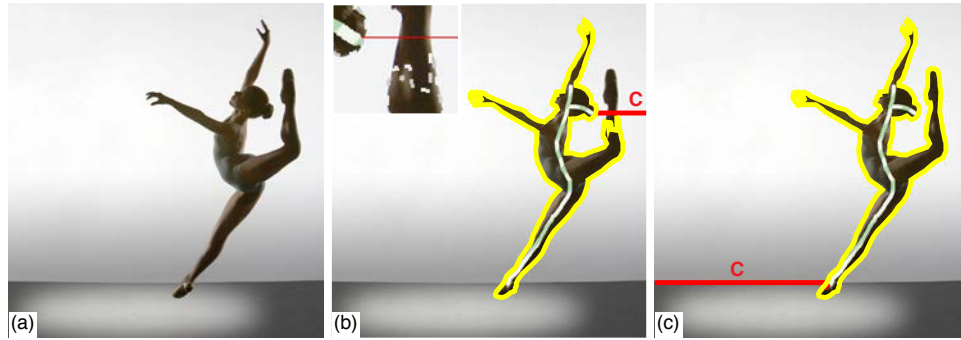


Figure 2: (a) Input image. (b) Drag-and-Drop Pasting [JSTS06] provides a boundary solution that fails to properly find the dancer's left leg (inset), due to Jia et al.'s minimum distance separating cut of the domain (red). (c) Our new formulation using a minimum energy separating path (red) is guaranteed to find the minimum boundary.

trees. Accordingly, the minimum boundary with the given constraints can be provided instantly and the constraints can be manipulated interactively. Recent work on panoramas [STP12] uses a variation on this approach to combine the ease of an automatic solution along with the direct, semi-automatic editing of the boundary. The previous work is limited to panoramas, since it assumes that the overlaps are small and that each image has unique areas outside the overlap. Therefore the technique of Summa et al. [STP12] is incapable of handling inset images. Our constraint interaction approach achieves the Summa et al. ease of use without limitations and their supported cases are simply a subset of the cases handled by our work. Li et al.'s Lazy Snapping [LSTS04] provides a polyline approximation of an automatically computed boundary for a user to adjust and constrain. After edits, additional automatic solutions are produced using the edited polyline as a guide. In contrast, a user can interactively explore the boundary solution space with our approach without the need for additional optimizations or approximations.

3. Paint and Click

The core of our algorithm consists of the following: painting annotations, finding the minimum cycle, and the addition of constraints.

Painting Annotation: To compute boundaries for an image, the user begins by labeling the pixels to *include* and/or *exclude* from the selection via a painting metaphor. Multiple, coincident, “like” painting strokes are computed as a single annotation. Similar to other boundary techniques, at least one annotation must be defined; otherwise the boundary problem is ill posed. Our base algorithm assumes that each annotation is independent and therefore, for the remainder of this section, we discuss solutions for a single annotation. In Section 3.4, we show how to intuitively combine the independent solutions.

Minimum Cycle: Our algorithm finds the minimum closed path that optimally separates the annotated pixels from the

rest of the input. We refer to this path as our *minimum cycle*. Our technique differs in two ways from the previous work [JSTS06]. (1) We provide a new solution to the minimum cycle calculation that is robust and guaranteed to find the minimum cycle. Figure 2 demonstrates where the previous work fails to properly segment the dancer. (2) Our algorithm has significantly less complexity than that of Jia et al. [JSTS06]. Our complexity provides a practical improvement of up to a 48 times speedup in our test data and allows our unified approach to be on par with the fastest paint-only boundary technique [STC09].

Adding Constraints: Our approach for calculating the minimum cycle enables the user to quickly and interactively add multiple constraints in order to refine the boundary while keeping it minimal. We perpetually stay one step ahead of the user with quick precalculations, thereby keeping all interactions fluid and instantaneous.

The next three sections will provide details on our algorithm and describe how our implementation enables a unified formulation. Section 3.1 will define the parameters of our input and basic definitions for the exposition of our technique. We will detail how to compute the minimum cycle in Section 3.2 and how to easily and efficiently add interactive user constraints in Section 3.3.

3.1. Pairwise Boundaries

For ease of explanation, we will assume that two overlapping images, A and B , serve as the input to our technique. In the case of object selection or foreground/background segmentation, A and B are the same image and the following formulation still applies. Given the two input images, we want to compute the pairwise boundary, boundaries between two images or an image with itself, such that there is a discrete labeling L for all pixels in their overlap. The labeling determines the image that contributes a pixel to the final composite or the pixel that is selected. The labeling is defined as $L(p) = \{A, B\}$ for location p in the overlap. Input for this procedure is typically an initial labeling provided by user

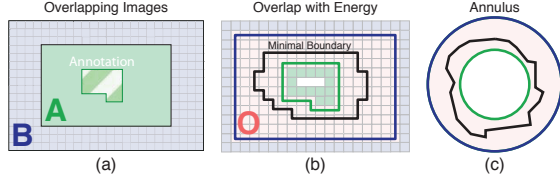


Figure 3: (a) Given a user annotation (green striped), a minimum boundary between images with complete overlap can be found by computing a minimum cycle. (b) and (c) illustrate O as the solution space given by the overlap of A and B , the boundary of overlap (blue) and the region annotated by the user (green), and the minimum cycle (black).

annotations, another algorithm, or a partial overlap. For purposes of illustrating such labeling in this paper’s figures, all *include* annotations are shown in striped green and *exclude* annotations are shown in red. Each annotation can have a single or multiple connected component(s).

The labeling can be computed by minimizing the transition [ADA*04], E_t , between images based on a piecewise smoothness $E_s(p, q)$, where $(p, q) \in \mathcal{N}$ and \mathcal{N} is the set of all neighboring pixels in the overlap.

$$E_t(L) = \sum_{(p,q) \in \mathcal{N}} E_s(p, q).$$

The smoothness energy can vary based on the type of transition required. For example, equations can minimize:

$$E_s(p, q) = \|I_{L(p)}(p) - I_{L(q)}(p)\| + \|I_{L(p)}(q) - I_{L(q)}(q)\|.$$

or maximize the transition in pixel values:

$$E_s(p, q) = e^{-\|I(p) - I(q)\|} \quad (1)$$

when $L(p) \neq L(q)$ and $E_s(p, q) = 0$ otherwise.

Given an initial labeling of the pixels, the problem is to find a new labeling that minimizes the transition between labels. Towards enabling the minimization, we represent the pixels as a planar, 4-neighborhood, energy-weighted pixel graph $G = (V, E)$, where V are pixel locations in the overlap and E are edges that connect pixel neighbors. Edges are weighted by the energy function E_s .

Most previous techniques would produce a new minimal labeling via minimum cut of the graph, such as in Li et al. [LSTS04]. As this previous work has shown, minimum cuts are insufficient for a fully unified technique since boundary edits require approximations and repetitive optimizations. In contrast, minimum paths have been shown to provide user control of the boundary without approximations [MB95, MB98, STP12], resulting in a what you see is what you get (WYSIWIG) interaction. In addition, Summa et al. [STP12] showed that a dual graph between pixels can be formed, on which the minimum path can produce the same solution as a minimum cut. We overcome the deficiencies of minimum cut boundary interactions and

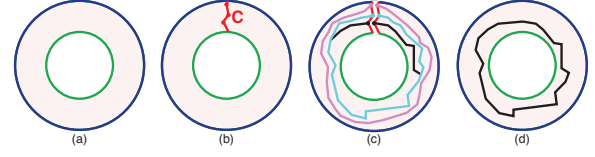


Figure 4: (a) Computation of a minimum cycle illustrated via a planar annulus. (b) A separating path is computed as the minimum path from interior (green) to exterior (blue) boundary nodes. The domain is split into a disk by replicating this path on either side of the domain. (c) If a set of minimum paths from the separating path nodes to their replicated selves is computed, the minimum cycle (d) is the minimum path in this set.

the limitations of the open paths of previous minimum path work [MB95, MB98, STP12] by providing a unified approach based on closed minimum paths.

As an added benefit, using minimum paths provides the option to operate and minimize on the boundary pixels themselves; an operation that is extremely useful for color correction applications. In particular, for seamless composition it is desirable to minimize the pixel color difference on the boundary:

$$E_t(L) = \sum_{p \in \partial L_i} \|I_A(p) - I_B(p)\|, \quad (2)$$

where ∂L_i is a set of points such that if $p \in \partial L_i$ then $L(p) = i$ and $L(q) \neq i$ for some q such that $(p, q) \in \mathcal{N}$ and i would correspond to the image that is to be seamlessly composed.

We assume in our exposition that the minimum boundaries occur on the pixels themselves. In other words, each boundary is a sequence of pixels (p_1, \dots, p_n) and the *nodes* of our graph are the pixel locations with edges connecting nodes corresponding to adjacent pixels.

3.2. Computing the Minimum Cycle

For clarity in the illustrations in the paper, we will describe the computation of the minimum cycle via the planar annulus, shown in Figure 3. A planar annulus represents the solution domain for a single annotation and is defined as the area between two concentric boundaries. As shown in Figure 3c, the interior boundary (green) of the annulus represents the annotation boundary and the exterior boundary (blue) represents the solution domain boundary. We will also describe a *separating path*, consisting of the pixels or *nodes* in the pixel graph that joins the interior and exterior boundaries.

Our approach for computing the minimum cycle makes two major improvements to the work of Jia et al. [JSTS06], including a robust separating path for splitting the planar annulus defined by the boundaries and a lower complexity divide-and-conquer approach for generating an optimal minimal cycle between the interior and exterior boundaries. Our

method results in a fast computation and the correct minimal cycle, as we will describe in this subsection.

3.2.1. Robust Separation of the Domain

In order to compute a minimum cycle between these boundary paths, we first create a separating path, C , that travels from the nodes of the exterior boundary to the interior boundary (Figure 4b). We then replicate the nodes of the separating path to split our solution domain. The minimum cycle is found via a search through the collection of all minimum paths from the separating path nodes, n_i , to their replicated selves, n'_i [IS79, Rei],

$$\text{MinCycle} = \min_{n_i} (\text{MinPath}(n_i, n'_i)).$$

Figures 4c and 4d illustrate the search for minimum cycle.

In our work, we make a critical adjustment to the minimum distance cut technique used by Jia et al. [JSTS06]. In the previous work, the separating cut was based upon the minimum distance from the interior to the exterior of the annulus, which does not account for the underlying energy. Jia et al. assumed that a minimum boundary would not cross the separating cut more than once, a safe assumption for simple color correction boundaries. However, when this assumption does not hold, such as in the case of a more general image segmentation or object selection, their technique will not find the minimum boundary. As shown in Figure 2b, a minimum distance cut bisects one of the legs of the dancer. Therefore, any boundary produced using such a separating cut cannot trace the inside this leg completely.

As shown in Figure 4, we compute our separating path as the minimum path on the underlying energy, not distance, between all nodes on the exterior boundary to the nodes on the interior boundary. The separating path can be computed by connecting the nodes of the exterior boundary with zero-weighted edges to a source dummy node and connecting all interior boundary nodes to a destination dummy node. The separating path is then computed as the minimum path between the source and destination nodes [IS79, Rei]. Since the separating path connects interior and exterior boundaries, the minimum cycle must cross it once and only once. Appendix A explains this property in detail.

3.2.2. Zero Constraints – Divide and Conquer

Additionally, our method improves upon the performance of the Jia et al. [JSTS06] boundary solution by introducing a new divide and conquer algorithm. Our algorithm reduces the runtime of our zero constraints solve from $O(MN)$ to $O(N \log M)$ for integer energy [MB95, MB98] and $O(MN \log N)$ to $O(N \log N \log M)$ for floating point energy [Dij59], where M is the length of the separating path and N is the number of pixels in the annulus. The complexity improvement provides as high as a 48 times speedup in our test data. The complexity of Jia et al.'s algorithm fits their desired offline boundary solution, but our goal is to produce image boundaries with

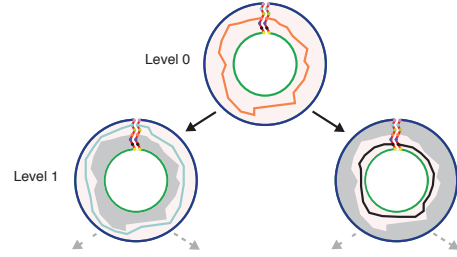


Figure 5: Our divide and conquer strategy for finding the minimum cycle first computes the cycle associated with the separating path's midpoint, the separating path is split about this midpoint, and then the algorithm recurses on the two subpaths. The solution domain in the recursion is partitioned by the midpoint's cycle with each partition including the cycle itself. Levels are dependent on previous levels, but operations of the same level are independent and parallel. Grey areas denote areas that are excluded from the computation.

interactive feedback. Moreover, the previous work assumed the separating path length, M , is small, an assumption that does not hold for our robust separating path in applications such as object selection.

Our recursive binary divide and conquer strategy, motivated by the work of Reif [Rei], exploits the fact that the minimum paths computed in finding the minimum cycle can be coincident but cannot cross (Appendix A). Figure 5 illustrates a step of our recursive algorithm. First, the minimum path from middle node of the current separating path is computed to its replicated self. The separating path then is split about the midpoint and the algorithm recurses on the two subpaths. The solution domain in the binary recursion can be partitioned by the midpoint's minimum path with each partition including the path itself.

3.3. Adding Interactive User Constraints

Our unified approach provides the ability to easily add user constraints since adding a single constraint is equivalent to finding the minimum cycle between the interior and exterior boundaries that must pass through a specific node. Adding additional constraints simply requires the examination of current constraints and their minimum path trees, as we will explain in the following subsections.

3.3.1. Single Constraints

Adding a single constraint includes building clockwise (CW) and counterclockwise (CCW) minimum path trees for all separating path nodes, finding the minimum cycle through the constraint, and creating a separating path through the constraint as a preprocess for additional constraints.

For every node of the separating path, we compute the minimum path tree in both clockwise and counterclockwise

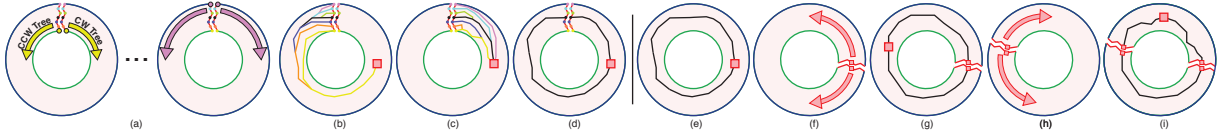


Figure 6: User constraints. (a) For every node in the separating path, a clockwise and counterclockwise minimum path tree is computed with the node and its replicated self as their roots. (b and c) To find a minimum cycle that passes through a constraint, one must simply find oriented paths associated with a node on the separating path which provides the minimum cost (d). (e) After the first constraint is added, the domain can be separated via the minimum path that passes through the constraint. (f) Two clockwise and counterclockwise oriented minimum path trees are computed with the constraint and its replicated self as their roots. (g) An additional constraint is simply a lookup on the two trees. (h) Each constraint has its own separated domain for computation. (i) New constraints are simple lookups on the constraint minimum path trees. The ordering of the constraints can be evaluated during any user interaction to allow for fluid movement of the constraints.

orientations with the node (or its replicated self) as the root. See Figure 6a. Each tree can be encoded as a step-direction and cost buffer. The step-direction buffer encodes the direction of a node's parent in one byte whereas the cost buffer stores the minimum path cost for each node at a desired precision. This preprocess has complexity a of $O(2NM)$.

We define the *constraint-minimum cycle* as the minimum cycle that must pass through the constraint(s). To find the constraint-minimum cycle, we find the oriented paths from a separating path node, n_i , and its replicated self, n'_i , to the constraint whose sum gives the minimum cost:

$$\text{MinCycle}(c) = \min_{n_i} (\text{MinPath}(n_i, c) + \text{MinPath}(n'_i, c)),$$

where c is the constraint location (Figures 6b, 6c, and 6d). The cost buffer can be dropped after computation by keeping track of the minimum cycle cost along with the index of the separating path node whose trees provide the cycle.

After the initial constraint is positioned (Figure 6e), a minimum path between the exterior and interior boundaries containing the constraint can be used to separate the annulus into a disk. We call this the *constraint-separating path*. As a preprocess to enable the addition of more constraints, two oriented minimum path trees (CW and CCW) with roots being the first constraint node and its replicated self (Figure 6f) are computed with a cost of $O(2N)$.

3.3.2. Two or More Constraints

Given the two oriented precomputed minimum path trees, finding the boundary with a second constraint is simply a tree traversal (Figure 6g). Additional constraints operate similarly, as illustrated in Figures 6h and 6i; after each constraint is set, a similar preprocess of generating oriented trees occurs for the next constraint.

Constraints are stored in a circular array and are inserted as follows. Let us assume we have three constraints, c_1 , c_2 , and c_3 , and wish to insert c_4 . The first three constraints are stored in a circular array as c_1 , c_2 , c_3 , c_1 . When the user adds c_4 , the algorithm evaluates the best place for c_4 by adding it between each pair of current constraints

$((c_1, c_2), (c_2, c_3), (c_3, c_1))$ and examining whether the result is the smallest cost boundary that encloses the annotation label. To find this, we simply compute the total cost for a boundary in all possible orientations. Multiple combinations of oriented paths are possible (4 in the standard case and up to 16 due to node replication if the new constraint is on the constraint-separating path of both constraints in the pair). The cost calculations are a simple lookup and the search rarely needs to test more than a few boundaries. The performance cost of this calculation is nominal in our testing.

Our scheme evaluates the constraint array position on both addition and movement allowing a user to add and move constraints instantly and fluidly without restriction. Because the minimum path can cross the constraint-separating path once and only once on a subpath containing the constraint (Appendix A), the boundary is guaranteed to be minimal under the constraints.

3.4. Multiple Annotations

Users often make multiple annotations in order to interactively add and carve pieces of an image until the desired boundary is found (see the supplemental video captures). The discussion of our technique up to this point considers only a single annotation. In this section, we detail our process for combining multiple annotations.

Given multiple boundaries, we create the pixel labeling for each by rasterizing the boundary geometry. We union *include* labels and remove *exclude* labels to form the selection using a hierarchical tree structure, similar to 2D constructive geometry, describing the nesting of labels. We will detail our scheme using the examples of Figure 7.

During the minimal cycle calculation and constraint manipulation, discussed in Subsections 3.2 and 3.3, our approach treats each connected component of the annotations as an independent, single annotation, but applies the other annotations as areas of high energy. The high energy areas guarantee a cycle does not split another annotation.

Figure 7a illustrates the case of an annotation that is not simply connected and lacks a different label in its interior.

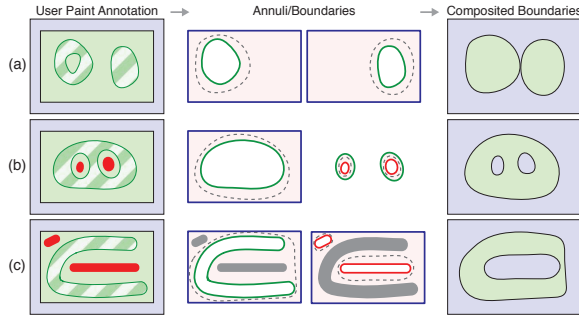


Figure 7: Combining minimum cycles for multiple annotations; include regions indicated with green stripes and exclude regions in red. (Row a) Our algorithm treats each annotation as independent and the solution as the union of like annotations. Non-simply connected regions without the presence of an opposite annotation are treated as simply connected. (Row b) Opposite annotations are treated independently and are subtracted from the boundary solution. The solution domain is reduced if an annotation is enclosed by a different annotation. (Row c) Combining annotations allows users to intuitively add/remove image regions.

In this case, the annotation is treated as simply connected and provides the proper solution. As in Figure 7b, if an annotation is enclosed by another we can restrict the solution domain to be only inside the enclosure. The combination of the different labels provides the final solution.

If a boundary is not nested, then it can be simply applied to the composite image because the boundary is a distinct partition of space, shown in Figure 7a. To combine nested boundaries, as seen in Figures 7b and 7c, we traverse the hierarchy bottom up and remove the each boundary's labeling from the labeling of the hierarchically lowest parent of the opposite label.

The boundary computation for applications with on-boundary optimizations have a subtle, yet important, distinction. In the final labeling, we retain the minimum boundaries when removing a labeling. For instance, the image to be color corrected in seamless composition should retain the minimum boundaries. Therefore, when removing the boundaries of the opposite label, the technique must not remove the labeling of the boundary itself in order to maintain the proper boundary.

Nested initial boundaries, shown in Figures 7b, cannot cross (Appendix A). However, for cases such as Figure 7c, the addition of user constraints may break this rule. To keep our scheme simple, we consider the crossing an unsupported state since it implies an ambiguous labeling and present the problem to a user. In practice, the unsupported state does not commonly occur.

To ensure proper computation of minimum cycle boundaries, we require at least one pixel space between opposite

annotations when optimizing on the boundary. When optimizing between pixels, we require a two pixel space.

Our new combination scheme provides good, intuitive results for a user by mimicking the natural adding and removing of image pieces with its one-to-one correspondence between annotations and the components/holes of the final selection.

4. Computation and Memory Improvements

In this section, we provide details on how to reduce the (C)omputational and/or (M)emory complexity with novel strategies or approximations. Our optimization methods include the use of hierarchical boundaries and improved initial constraint initialization, including discussions on automatically setting the initial constraint(s), tree computation halting, cycle compression to reduce storage, and a subsampled separating path approximation.

4.1. Hierarchical Boundaries - (C, M)

We provide a hierarchical version of our algorithm to help both computational performance and memory overhead. Our unified approach is easily adapted to a hierarchical solution with the following steps: find the minimum boundary, up-sample and dilate the minimum boundary, and use the dilation as the new domain for the next level of the hierarchy. A dilation of a minimum boundary is typically another annulus. The exception is when the dilated domain causes a pinch or connection between pieces of the boundary that are close in distance but not close on the boundary itself. This exception can be avoided by thresholding the dilation based upon an on-boundary distance to allow close boundary regions to connect and far regions to remain disconnected. In practice, we used dilations in the range of 5 to 30 pixels.

4.2. Improved Initial Constraint Preprocess

The $O(2NM)$ step in Figure 6a and discussed in Section 3.3.1 can quickly become a performance and/or memory bottleneck. The initial constraint step can be accelerated, the trees can be compressed, and approximate solutions can be produced that are still of high quality.

Tree Computation Halting - (C) We have observed that for each oriented tree in Figure 6a, there is little difference in structure between the trees of the same orientation as they get further away from the root. Therefore, we have devised a strategy that has given us a 30% to 60% reduction in computation in our test data. First, an oriented, *base* tree from a node in our separating path is computed along with a collection of rastered lines from the interior to exterior boundaries that partition the annulus into regions. We refer to each raster line as a set of *test nodes*. During the computation of a new tree, for each set of test nodes if each node in the

set has a cost that differs from the cost in the base tree by a common Δ , then we can treat the paths for all nodes beyond the test nodes as equivalent and the computation can halt. Appendix B provides an informal proof. In our work, we found that 32 raster lines provides the best performance in the trade-off between amount of testing and halt locations.

Cycle Compression - (M) Two properties of the minimum cycle lead to a simple storage reduction for a large number of trees and follow directly from the property discussed in Appendix A. First, a constraint-minimum cycle that touches the minimum cycle cannot cross the minimum cycle and the only subpath that is not coincident is the subpath that contains the constraint itself. Therefore, to encode a constraint-minimum cycle that touches the minimum cycle, all we need to save is the minimum cycle and this connecting subpath. Second, if we consider this subpath connecting a constraint to the minimum cycle as having two components defined by its orientation, all like-oriented subpaths from constraints to the global minimum cycle cannot cross or separate when coincidence occurs.

Given both properties, all that is needed to encode the constraint-minimum cycles that touch the minimum cycle are two buffers to store the clockwise and counterclockwise paths. For all constraints that define these cycles, only the first step in either orientation is needed to encode its path. Minimum cycles for each constraint can be built by walking each buffer from the starting constraint and terminating upon finding a common node. Adversarial cases can be constructed with poor compression, but we have found these are rare in practice. In fact, practically all constraint-minimum cycles in our test data touch the minimum boundary, thereby leading to very high compression for the oriented tree buffers.

Automatic Initial Constraint - (C, M) We can bypass the computation of Section 3.3.1 (Figure 6a), skipping directly to Section 3.3.2 (Figure 6e), by automatically inserting constraints on the minimum cycle in one of two ways. The first option picks a point on the minimum cycle to add to the boundary as a constraint. The constraint-minimum cycle for points on the minimum cycle is the minimum cycle itself. The two oriented trees for the constraint are computed, $O(2N)$, instead of the $O(2MN)$ constraint preprocess. A user can then add and move a second constraint per the usual algorithm. The first constraint cannot be moved until the second constraint is added and positioned. To avoid this locking, the second option adds two points on the minimum cycle as constraints. In this approach, four trees are computed, $O(4N)$, but both constraints are editable from the start.

Subsampled Separating Path Approximation - (C, M) The computation of trees in Figure 6a does not necessarily need to be a full sampling of the separating path to produce results of high quality. In practice, we have noted that



Figure 8: (a) The coarse painting of the eagle selects the eagle's body. The fine features of the talons and feathers are selected with simple clicks, which avoids tedious pixel-wide painting (inset purple). (b) The coarse annotation of the snowboarder selects his body and board. The user-added constraints refine fine features such as his hand and fix other ambiguous regions on his body, avoiding pixel-wide painting (inset purple). (c) A coarse painting selects the statue and the boundary is fixed with constraints.

subsampling the path with every other or every third node produces approximate constraint-minimum cycle results imperceptible to the user from the actual minimum.

5. Results

In this section and the accompanying video, we show our new unified interaction for examples in object selection and seamless composition. All timings were performed on an i7 3.5 GHz desktop using wall clock time. For illustration, constraints and boundaries have been coarsely outlined.

Object Segmentation/Selection. Figures 1 and 8, along with the companion video, provide examples of our technique being used for interactive object selection. Specifically, the energy from Equation 1 is minimized on the dual of the 4-neighbor pixel graph. For this application, we have found no hierarchy was necessary for good performance and approximating by subsampling the separating path every third node reduced the preprocessing time for the initial constraint sufficiently while maintaining quality results.

Figure 1 illustrates the benefits of our algorithm in select-

ing a beetle on a leaf. Figure 1c shows the strokes required to select the beetle using Adobe Photoshop CS5's quick selection tool, which was recorded via macro. The quick selection tool is an example of a purely painting approach. Figure 1d demonstrates the clicks required (red) and automatic anchors (yellow) set with Photoshop's magnetic lasso tool; a pure constraints approach. As the figure and the supplemental video show, there can be many tedious interactions necessary to get the desired result in these non-unified settings. Using our unified environment (Figure 1e), a user can avoid the careful painting of the antennae and legs and the many clicks required for constraints on the beetle's body. In addition, the figure also shows how few constraints are needed since they are only refining an already well-placed initial boundary provided by the painting. In our supplemental videos, we provide an example of the significant time savings granted by our new approach when compared to pure painting and pure constraints. We have tested our prototype with several experienced photo editors and have received unanimously positive feedback on the quickness and intuitiveness of our unified approach. All users would like to use this tool in an interactive photo editing suite.

Like the previous example, often the fastest selection is the result of an initial, quick coarse painting that is refined with constraints. In Figure 8a, a user selects an eagle with easy broad painting strokes. The feathers and talons would be tedious to paint; therefore, a user clicks and adds a few constraints to adjust the boundary for the talons and the wings. As the purple inset images show, these constraints avoid careful pixel-wide paintings. In Figure 8b, a user selects a snowboarder with a quick *include* (green striped) and *exclude* (red) stroke. Like the eagle, the athlete's right fingers would be difficult to paint. They can be selected with a few clicks. In addition, there are areas the initial painting missed such as his foot and left hand. Either additional painting annotations could be added, or as this example shows, a few clicks can fix the selection. In Figure 8c, multiple paintings are applied to the image of the statue. The initial result produces a boundary that includes the background building and sky, while excluding pieces of the statue. These problems can be fixed with just a handful of constraints.

Seamless Composition. Another application of our technique is the creation of a seamless composition. In this application, the boundaries are combined with a color correction technique such as gradient domain blending [PGB03, LZPW04] to produce a seamless image. The foreground image is seamlessly blended into the background by matching the foreground boundary's pixels to the background and solving a Poisson system to blend the color difference into the foreground's interior. A logical foreground boundary would be one that deviates from the background as little as possible. In particular, we use the energy from Equation 2 on the 4-neighborhood pixel graph. The work of Jia et al. [JSTS06] targets both offline boundary and color cor-

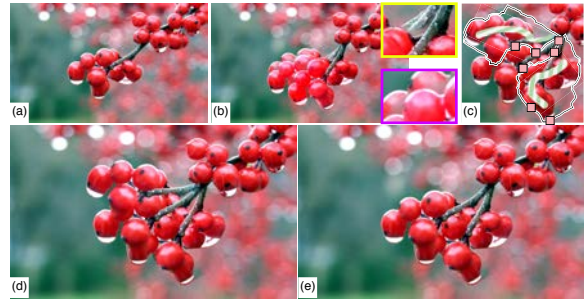


Figure 9: (a) An image of berries where the bottom branch will be cloned multiple times. (b) The result from Farbman et al. [FHL*09]. Even the most advanced color correction techniques can be limited by the chosen boundaries. Examples of problems include disappearing branches (inset yellow) and color inconsistencies due to inclusion of the background in the color solution (inset purple). (c) With our unified interactions, boundaries can be guided by the user to give the best color result. (d) The final image with two cloned branches. (e) An image with one cloned branch.



Figure 10: A user manipulates the moose's antler and eyes. (a) The original image. (b) Eye size is increased and antler size decreased. Problems such as the purple inset are instantly detected by a user with our real-time boundaries. (c) Eye and antler size increased. Boundaries and interactions used to produce the seamless image are white inset.

rection, but in our work we have focused on providing interactive boundaries with quick color correction after manipulation. The boundary interaction is important since problems in the final composition can be easily caused by boundaries. A minimum boundary may intersect with a very distinct part of the scene, leading to bad color correction and a poor final composition. Figure 9b shows a result from Farbman et al. [FHL*09], who provide a high quality color correction routine. As the inset images show, the color correction can only do so much to provide a quality image when the boundaries are not set realistically. Examples in our companion video, such as the berries example, demonstrate instances of improper boundaries and illustrate of how to correct them with our system. Our software allows users to see and/or edit boundaries while manipulating the foreground images. In addition, users can edit boundaries and suppress the color correction to preserve hard edges, for example, the top of the fins in orca of Figure 11. Therefore, boundary editing and color correction can work in tandem to provide the best quality image. Finally, since our target is pairwise bound-

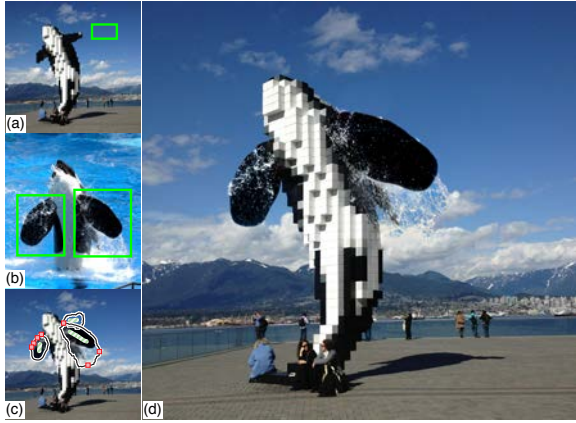


Figure 11: (a and b) Combining two images to blend abstract and real. The areas bounded by green are seamlessly composited into the first image. (c) Our unified interactions are used to create a composite (d).

aries, multiple foreground images are handled by processing each sequentially.

Figure 9a illustrates an image where the berries are cloned to make a larger cluster. With our technique, the image pieces can be manipulated and moved with boundaries edited by our unified approach (Figure 9c). The resulting final images, Figures 9d and 9e, provide a more realistic color correction than the bright red berries of Figure 9b. In Figure 10, a user clones and manipulates a moose's antlers and eyes to make two different scenes (Figures 10b and 10c). Figure 10b illustrates how the real-time boundaries are fundamental in positioning the antlers realistically while simultaneously getting a sense where problem areas in the boundary may break realism (purple inset). Figure 11 provides a more complicated example of seamless cloning and compositing to produce a meld of the orca statue, Figure 11a, and a photograph of a real orca, Figure 11b. Composited portions are highlighted in green. With our unified interaction, Figure 11c, we can produce a quality final image, Figure 11d.

Performance. The performance of our technique can be evaluated in several ways. We first compare the object selection performance for our zero constraints solution against the fastest Graph Cuts implementation for planar graphs [STC09]. Additionally, we provide the running times for our techniques with the various acceleration strategies presented in Section 4. A more complete tally of our timing results is provided in the supplemental material. Overall, with our acceleration strategies our approach is interactive for the initial boundary solution and provides instantaneous feedback while a user edits constraints.

Our technique for image segmentation may have slightly higher complexity than the previous work [STC09], but has comparable running times on our 4-core machine. Since our technique has parallel elements, it will improve as more

cores are added. The beetle in Figure 1 with 715x1023 pixels took 0.66s in the planar Graph Cuts technique and 0.67s with our zero constraints solution. The images in Figure 8 were 0.36s, -0.08s, 0.9s faster with our technique for images that range between 984K and 2.5M pixels. Additionally, the average cost after addition or movement of a constraint, as discussed in Section 3.3.2 (Figure 6f), taken from a typical editing session ranges between 158ms for a 490K pixel image to about 798ms for a 2.5M pixel image.

The previous timings do not use hierarchical acceleration. We now discuss the effect of the hierarchy for the color blending example for the left fin shown in Figure 11. With no hierarchy, the zero constraints solution takes 701ms for a full resolution solve. The single constraint preprocess takes 5.1s with subsampling the separating path every third node. Although fast, the image processing does not yet achieve the our desired interactivity. Using two levels of hierarchy, the zero constraints solution takes 151ms and the single constraint preprocess takes 0.58s. Three levels reduce the time further to 37ms for the zero constraints solution and 0.07s for the single constraint preprocess. Resolving the full resolution solution via three levels of the hierarchical solver requires 33ms or 112ms depending upon the dilation (5 and 30 pixels, respectively). We have found, in practice, that 5 pixel-dilation allows a user to produce a full solution visually close to the coarse solution whereas 30 will produce the same solution as no hierarchy. For two levels, the solution takes 29ms for 5 pixel-dilation and 84ms for 30 pixel-dilation. In all cases, the full solution can be quickly provided and the cost of the hierarchical solution is nominal. In our seamless cloning examples, the hierarchy provides interactive rates allowing the manipulation the foreground image and boundaries in real-time.

6. Discussion and Limitations

Although a complete overlap was the target application for our unified approach, Figure 12a illustrates how our approach can be easily used for partial overlaps between two images. This case is commonly seen in panoramas. If we connect each intersection point to a dummy node with zero-weighted edge and connect, round-robin, pairs of dummy nodes with zero-weighted edges, we form a solution domain that is equivalent to a collapsed annulus. In Figure 12b, the dashed line is the collapsed region. Note that the boundaries that contain o_1 or o_2 can be considered the interior and exterior boundaries, or vice versa. Either choice would produce the same solution. Odd numbers of intersections and raster artifacts can be handled as specified in the algorithm of Summa et al. [STP12]

Our unified interaction targets pairwise image boundaries. For more than two images, painting approaches using algorithms such as Graph Cuts provide the most common solution. Interactive constraints for more than two images work only in specialized applications [STP12]. Seamless compo-

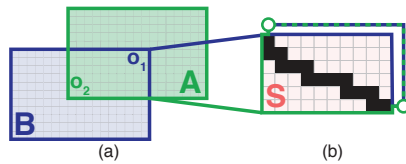


Figure 12: (a) Boundaries for partial overlaps are supported by our approach by connecting nodes of image boundary interactions to dummy nodes. (b) Connecting, round-robin, pairs of dummy nodes with zero-weighted edges, we form a solution domain that is equivalent to a collapsed annulus.

sitions of multiple images or photomontages [ADA*04] can be constructed with our technique by sequentially composing each image into the background. We believe that the new unified interaction provides good results in these cases, but we acknowledge this may not always be the case. If users want a minimum boundary with more than two images without any interaction beyond painting, then techniques such as Graph Cuts are the best option. However, iterative techniques such as Graph Cuts are prone to local minima and use pairwise boundaries as the core of their optimizations. We believe our approach can aid these techniques by allowing user interaction to avoid these suboptimal states.

Minimum cut algorithms allow for a data energy term applied per pixel for a given label. Future extensions of our technique will deal with how to integrate such data costs. Due to our pixel 4-neighborhood, in areas of smooth energy the path may take a "manhattan" walk rather than an equivalent straight walk depending on the order neighbor pixels are traversed in the optimization. We have found that this effect is noticeable only in contrived examples. As outlined in the text, after a constraint is added, movement is possible before its oriented trees are computed, but we have found, in practice, a user often adds several constraints at once in the paint-and-click model. Therefore, in our prototype we have chosen to compute a constraint's oriented trees as soon as it is added. Although precomputing the trees adds a delay, it keeps the system intuitive for users.

7. Conclusion

We provide a novel, unified approach for interactive editing of image boundaries that combines the ease of painting with the direct control of constraints. Our zero constraints boundary based on a painting annotation is faster and more robust than the previous work and is on par with the performance of the best paint-only solvers. A user can add multiple constraints and instantly edit the boundary. Our interactive user flow allows for the full exploration of the solution space, even at a fine level. Our unified approach leads to more flexible and faster editing sessions for users. In addition, we have shown how performance can be improved with novel strategies and approximations. Finally, we have provided exam-

ples of the real and immediate impact our new strategy has in digital photography applications.

References

- [ADA*04] AGARWALA A., DONTCHEVA M., AGRAWALA M., DRUCKER S. M., COLBURN A., CURLESS B., SALESIN D., COHEN M. F.: Interactive digital photomontage. *ACM Trans. Graph* 23, 3 (2004), 294–302. 4, 11
- [BK04] BOYKOV Y., KOLMOGOROV V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Anal. Mach. Intell* 26, 9 (2004), 1124–1137. 2
- [BVZ01] BOYKOV Y. Y., VEKSLER O., ZABIH R.: Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Analysis and Machine Intelligence* 23, 11 (2001), 1222–1239. 2
- [CSD03] COHEN M. F., SHADE J., HILLER S., DEUSSEN O.: Wang tiles for image and texture generation. *ACM Trans. Graph* 22, 3 (2003), 287–294. 2
- [Dav98] DAVIS J. E.: Mosaics of scenes with moving objects. In *CVPR* (1998), pp. 354–360. 2
- [Dij59] DIJKSTRA E. W.: A note on two problems in connexion with graphs. *Numerische Mathematik* 1 (1959), 269–271. 5
- [EF01] EFROS A., FREEMAN W.: Image quilting for texture synthesis and transfer. In *SIGGRAPH* (2001), pp. 341–346. 2
- [FHL*09] FARBMAN Z., HOFFER G., LIPMAN Y., COHEN-OR D., LISCHINSKI D.: Coordinates for instant image cloning. *ACM Trans. Graph* 28, 3 (2009). 9
- [IS79] ITAI A., SHILOACH Y.: Maximum flow in planar networks. *SIAM Journal on Computing* 8, 2 (1979), 135–150. 5
- [JSTS06] JIA J., SUN J., TANG C.-K., SHUM H.-Y.: Drag-and-drop pasting. *ACM Transactions on Graphics* 25, 3 (July 2006), 631–637. 2, 3, 4, 5, 9
- [KZ04] KOLMOGOROV V., ZABIH R.: What energy functions can be minimized via graph cuts? *IEEE Trans. Pattern Anal. Mach. Intell* 26, 2 (2004), 147–159. 2
- [LS10] LIU J., SUN J.: Parallel graph-cuts by adaptive bottom-up merging. In *CVPR* (2010), IEEE, pp. 2181–2188. 2
- [LSGX05] LOMBAERT H., SUN Y. Y., GRADY L., XU C. Y.: A multilevel banded graph cuts method for fast image segmentation. In *ICCV* (2005), pp. I: 259–265. 2
- [LSS09] LIU J., SUN J., SHUM H.-Y.: Paint selection. *ACM Transactions on Graphics* 28, 3 (Aug. 2009), 69:1–69:?? 2
- [LSTS04] LI Y., SUN J., TANG C.-K., SHUM H.-Y.: Lazy snapping. *ACM Trans. Graph* 23, 3 (2004), 303–308. 2, 3, 4
- [LZPW04] LEVIN A., ZOMET A., PELEG S., WEISS Y.: Seamless image stitching in the gradient domain. In *ECCV* (2004), pp. Vol IV: 377–389. 2, 9
- [MB95] MORTENSEN E. N., BARRETT W. A.: Intelligent scissors for image composition. In *SIGGRAPH* (1995), pp. 191–198. 2, 4, 5
- [MB98] MORTENSEN E. N., BARRETT W. A.: Interactive segmentation with intelligent scissors. *Graphical models and image processing: GMIP 60* (1998). 2, 4, 5
- [NFK07] NAGAHASHI T., FUJIYOSHI H., KANADE T.: Image segmentation using iterated graph cuts based on multi-scale smoothing. In *ACCV* (2007), pp. II: 806–816. 2
- [PGB03] PÉREZ P., GANGNET M., BLAKE A.: Poisson image editing. *ACM Trans. Graph* 22, 3 (2003), 313–318. 9

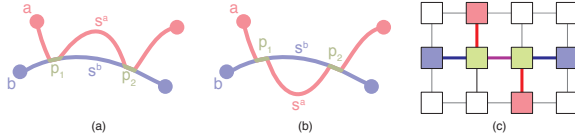


Figure 13: (a and b) Two minimum paths can be coincident only on a single connected subpath. More than one contradicts minimality if all path distances are assumed to be unique. (c) On a planar graph, two paths (red and blue) must touch or cross only on a connected subpath of both paths (green).

- [Rei] REIF J. H.: Minimum $s-t$ cut of a planar undirected network in $O(n \log^2(n))$ time. *SIAM J. Comput.* 12, 1, 71–81. 5
- [RKB04] ROTHER C., KOLMOGOROV V., BLAKE A.: Grabcut: interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph* 23, 3 (2004), 309–314. 2
- [STC09] SCHMIDT F. R., TOPPE E., CREMERS D.: Efficient planar graph cuts with applications in computer vision. In *CVPR* (2009), pp. 351–356. 3, 10
- [STP12] SUMMA B., TIERNY J., PASCUCI V.: Panorama weaving: fast and flexible seam processing. *ACM Trans. Graph.* 31, 4 (July 2012), 83:1–83:11. 2, 3, 4, 10
- [TGVB13] TANG M., GORELICK L., VEKSLER O., BOYKOV Y.: Grabcut in one cut. In *ICCV 2013* (2013), pp. 1769–1776. 2
- [VN08] VINEET V., NARAYANAN P. J.: CUDA cuts: Fast graph cuts on the GPU. In *Computer Vision on GPU* (2008), pp. 1–8. 2

Appendix A: Minimum Path Property

A detailed presentation of a property of minimum paths (cycles), particularly important for much of our work, is as follows. We assume all (sub)path lengths of a minimum path are unique and that our pixel graph is a planar 4-neighborhood graph. The unique path property is provided by enforcing a strict ordering of pixel preferences during minimum path calculations or geometrically collapsing equal subpaths since equal subpaths are equivalent solutions.

Property A.1 Any combination of two minimum paths can be coincident only on a single connected subpath. In Figures 13a and 13b, two paths (a and b) have coincidence on more than one connected subpath (p_1 and p_2). For the coincident subpaths to be disconnected, there must be distinct subpaths s^a and s^b connecting them from paths a and b, respectively. Since all path distances are assumed to be unique, the existence of s^a and s^b contradicts the minimality of a and/or b.

This property implies: (1) once a path becomes coincident with another, it remains that way except for the two subpaths that travel to its endpoints and (2) since all crossings between paths must occur on a connected subpath of both (Figure 13c), two paths or cycles cannot cross more than once.

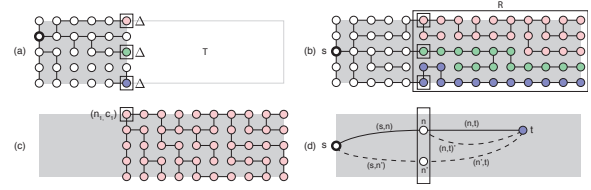


Figure 14: (a) Our tree halting scheme: when given a base tree T , if a tree computed differs from the base by a Δ at the cut nodes, computation can stop. The new tree in R is equivalent to T and node distances are T 's distance plus Δ . (b) A minimum path tree rooted at s and a set of nodes that cuts the domain R from s . (c) A minimum path tree, T_1^* over R from a cut node n_1 with the root having initial cost c_1 . (d) The dashed counter examples to Property B.1. Their existence contradicts the minimum tree assumption.

Appendix B: Halting of Minimum Path Trees

Consider our tree halting scheme of Section 4.2 and Figure 14a. The computation for minimum path trees can be halted for the following reason: Given a minimum path tree T computed from a root node s , consider a set C of nodes (n_1, \dots, n_k) with costs (c_1, \dots, c_k) on the boundary of a region R such that all minimum paths from nodes in R must pass through a node in C exactly once (Figure 14b). Every node, n_i , in C has a minimum path tree T_i with the node as root. A tree T_i^* is the minimum path tree with n_i as root, but where n_i is given cost c_i instead of zero (Figure 14c). It is easy to see that the two trees, T_i and T_i^* , are equivalent and that every node q in R has a cost: $\text{cost}(q, T_i^*) = c_i + \text{cost}(q, T_i)$. Given these definitions, we would like to show the following property:

Property B.1 For every node q in R , $\text{cost}(q, T) = \min(\text{cost}(q, T_i^*))$. There is a path in T from s to q that goes through some node $n_i \in C$, which defines $\text{cost}(q, T)$. If $\text{cost}(q, T_i^*)$ is lower, an alternative path would exist from q to n_i , which contradicts that T is a minimum path tree. Similarly, the cost cannot be higher. If it were higher, T_i^* would not be a minimum tree. For any other path, from q to a node $n_j \in C, i \neq j$, the cost of q with respect to T_j^* is greater than the cost of q in T . Otherwise this would imply an alternative, cheaper path through n_j , which contradicts T as the minimum tree. See Figure 14d.

Whereas this property deals with costs, it implies that each subtree of T rooted in n_i is a subtree of T_i^* . Scaling or translation/scaling are also viable cost differences, but we have found, in practice, that offsets provide good results and are cheap to compute. In addition, C is actually a subset of each of our raster lines. We overtest in order to have a simple procedure.

1. Additional Examples

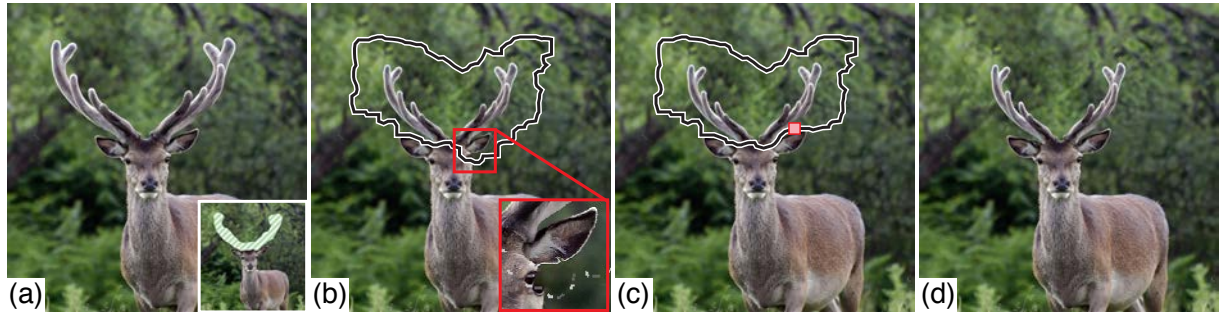


Figure A: A user manipulates the deer's antlers. (a) The original image with the initial painting interaction to select and clone the antlers. (b) The scaled antlers are placed in a realistic location in the scene. Minimal boundaries are provided interactively which the user can use to detect unrealistic transitions in the composite. (c) A single constraint can fix the bad transition to give the final seamless image (d).



Figure B: (a and b) Combining two images of motorcyclists to create more congested traffic. (c) The user paints the foreground motorcycle to composite into the other image. The initial solution has problems such as the passenger being excluded and unrealistic shadow transitions. (d and e) A user can simply add a few constraints to fix these issues.



Figure C: (top left) Several snapshots of a family composited into a single montage. Starting with a user-chosen background, pieces of three other images selected by the a painting annotation are composited into a seamless image. (top right) The initial boundaries have inconsistent transitions or undesirable areas included from an image, such as the child turning his head. With a few constraints (bottom left), all issues are resolved to produce a seamless image (bottom right).

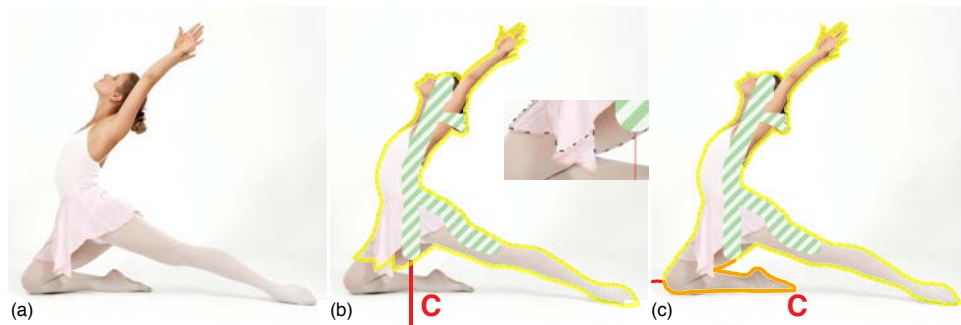


Figure D: Alternative to Figure 2: (a) Input image. (b) Drag-and-Drop Pasting [JSTS06] provides a boundary solution that fails to properly find the dancer's left leg (inset), due to Jia et al.'s minimum distance separating cut of the domain (red). (c) Our new formulation using a minimum energy separating path (red) is guaranteed to find the minimum boundary.

2. Performance

Figure	Size (Annulus)	Planar Graph Cuts	Our Method	Separating Path	Constraint Preprocess	Per Constraint
Beetle (Figure 1e)	715x1013 (490K)	0.66 s	0.67 s	368	2.1 s	158 ms
Eagle (Figure 9a)	1024x1024 (984K)	1.18 s	0.82 s	59	4.2 s	370 ms
Snowboarder (Figure 9b)	1900x1452 (2.5M)	2.44 s	2.52 s	68	8.8 s	798 ms
LOVE (Figure 9c)	1623x1259 (1.6M)	2.07 s	1.17 s	60	4.3 s	697 ms

Table A: Object selection timings for Planar Graph Cuts[STC09] (P-GC) and our minimum cycle computation on a 4-core machine. Actual annulus pixels are provided. We additionally provide the separating path size and preprocessing time for a single constraint without subsampling approximation (Figure 6a of paper), along with average cost after movement for constraints (Figure 6f of paper).

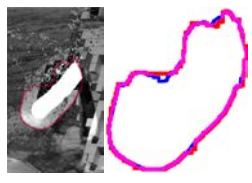
	Hierarchy Levels (Dilation)	Minimum Cycle	Constraint Preprocess (with Subsampling Approx.)	Hierarchical Solution
	1	701ms	14.7(5.1)s	-
	2(5)	151ms	1.7(0.58)s	29ms
	2(30)	-	-	84ms
	3(5)	37ms	0.19(0.07)s	33ms
	3(30)	-	-	112ms

Table B: The first image provides an example energy labeling of the left fin of the orca from Figure 11 in its final position (721x1234). The next image illustrates the expanded boundaries with a coarse solution for a 3-hierarchy (magenta) and a full solution (blue). As shown in the table, the hierarchy performs at interactive rates providing a user the boundary manipulating the image. Dilation results are provided for 5 and 30 pixels. The former provides a solution close to the coarse boundary, where as the latter produces the exact minimum boundary. For all cases, the cost of the final hierarchical solution is nominal.