Efficient Affinity-based Edit Propagation using K-D Tree

Kun Xu[1]   Yong Li[1]   Tao Ju[2]   Shi-Min Hu[1]   Tian-Qiang Liu[1]
[1] Tsinghua National Laboratory for Information Science and Technology
and Department of Computer Science and Technology, Tsinghua University
[2] Washington University in St. Louis

user strokes   (a) frame 180 **original video** (b) frame 240   (c) frame 180 **result video** (d) frame 240
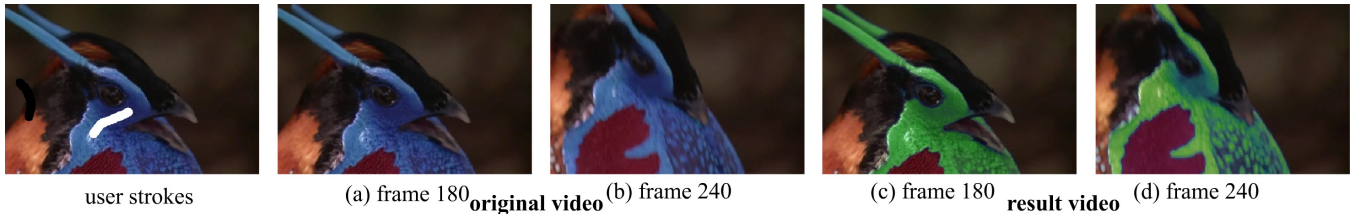
Figure 1: Affinity-based edit propagation methods such as [An and Pellacini 2008] allow one to change the appearance of an image or video (e.g., the color of the bird here) using only a few strokes, yet consuming prohibitive amount of time and memory for large data (e.g., 48 minutes and 23GB for this video containing 61M pixels). Our approximation scheme drastically reduces the cost of edit propagation methods (to 8 seconds and 22MB in this example) by exploring adaptive clustering in the affinity space. Video courtesy of BBC Motion Gallery (UK).

Abstract

Image/video editing by strokes has become increasingly popular due to the ease of interaction. Propagating the user inputs to the rest of the image/video, however, is often time and memory consuming especially for large data. We propose here an efficient scheme that allows affinity-based edit propagation to be computed on data containing tens of millions of pixels at interactive rate (in matter of seconds). The key in our scheme is a novel means for approximately solving the optimization problem involved in edit propagation, using adaptive clustering in a high-dimensional, affinity space. Our approximation significantly reduces the cost of existing affinity-based propagation methods while maintaining visual fidelity, and enables interactive stroke-based editing even on high resolution images and long video sequences using commodity computers.

1   Introduction

The need to edit the appearances of images and videos, such as their color, brightness and contrast, continues to rise. While traditional image editing environment, such as Photoshop, requires the user to first select the region of interest then apply the edit to that region, selection of intricate patterns is a laborious process, especially in videos where the region extends a sequence of frames (e.g., the bird in Figure 1) . Alternatively, recent methods [Lischinski et al. 2006; Pellacini and Lawrence 2007; An and Pellacini 2008] have allowed for a much simpler mode of interaction where the user only needs to supply edits to a sparse set of image locations (e.g., such as the strokes in Figure 1 far left). These methods then *propagate* the user edits automatically to the rest of the image without the need for accurate region selection, guided by the principle that image pixels with high *affinities* (e.g., similar appearances and/or at close proximity) should receive similar edits.

While propagation offers an intuitive editing mode, existing propagation methods often require prohibitive amount of memory and long time to process large inputs. As an example, edit propagation on the 400-frames video in Figure 1 using the method of [An and Pellacini 2008] requires 23GB of memory space, which would not fit in a commodity computer, whereas an out-of-core implementation requires 48 minutes of processing time. This high cost arises from the need, which is common in edit propagation, to solve an optimization problem whose size is proportional to the number of pixels in the data. The storage requirement and long user-waiting time significantly limit the practical use of edit propagation for high resolution images and long video sequences, which have become increasingly popular.

In this paper, we propose a novel, efficient paradigm for solving the optimization problem in edit propagation. Our key observation is that, since propagation yields similar edits at pixels with high affinity, the propagated edits form a smooth function, not in the image/video space, but in a higher-dimensional affinity space. As a result, we approximate this function by piece-wise linear segments in the affinity space, each represented by a cluster of nearby pixels. To satisfy the user-specified edits, the size of the clusters is adapted to their distances to the user-edited pixels. Instead of solving the optimization directly on individual pixels, as done in previous methods, we solve on the clusters, which is of a much smaller number (typically hundreds to thousands of times smaller) than the number of pixels. We have observed that our approximation preserves the visual fidelity of the original propagation method while using only an extremely small fraction of resources. For example, edit propagation in Figure 1 is done using our approximation scheme, which now takes only 8 seconds and uses 22MB of memory.

Contribution In the context of previous work on propagation-based image editing, we see our paper making two notable contributions, both in algorithm development and in practice:

- We propose an efficient approximation scheme for affinity-based propagation methods by exploring adaptive clustering in the affinity space, which reduces both memory usage and time significantly without sacrificing visual fidelity.

- Our approximation scheme significantly increases the practicality of edit propagation to handle large data, making it possible for average users to edit high resolution images and long video sequences on a commodity computer interactively.

2   Related works

Image/Video appearance editing: With the advance of appearance acquisition techniques, editing captured digital contents receives growing attention in recent years. Yatziv et. al. [2006] proposed an efficient colorization method for images and videos

using fast chrominance blending. To enhance the edit result in the presence of strong image edges, edge-aware methods [Chen et al. 2007; Li et al. 2008; Fattal 2009; Fattal et al. 2009] have been proposed for various editing tasks. In a different approach, optimization-based methods have been particularly successful in propagating rough, sparse user-edits to large, complex, and possibly spatially disconnected regions in images or videos. Example of such methods include colorization [Levin et al. 2004], tone adjustment [Lischinski et al. 2006] and material editing [Pellacini and Lawrence 2007; An and Pellacini 2008].

Hierarchical and clustering: Hierarchical and clustering techniques have been widely used in computer graphics. Well-known applications include radiosity [Hanrahan et al. 1991], the lightcuts rendering framework [Walter et al. 2005; Walter et al. 2006], lighting/material design [Cheslack-Postava et al. 2008] and image segmentation [Shi and Malik 2000]. Recently, [Agarwala 2007] proposed an efficient gradient-domain image compositing method using quadtrees, which largely reduces the dimension of the linear system to solve. Differently, we build k-d tree in a high dimensional space instead of in image space. For a different purpose, [Adams et al. 2009] proposed Gaussian k-d trees to accelerate a broad class of non-linear filters. Using a non-hierarchical approach, joint bilateral upsampling [Kopf et al. 2007] speeds up edge aware image adjustment by upsampling a solution at a low resolution.

3 Affinity-based edit propagation

Stroke-based image and video editing has become popular in recent years [Lischinski et al. 2006; Pellacini and Lawrence 2007; An and Pellacini 2008]. This type of editing only requires the user to draw a set of sparse strokes indicating the desired changes in appearance (e.g., color, brightness, contrast, etc.), such as the two strokes in Figure 2 (a), and automatically propagates these initial edits to the rest of the image or video. Typically, the propagation is guided by two principles. First, pixels that are "similar" (e.g., located closed-by, having similar colors, etc.) should receive similar edits. Second, pixels covered by the initial strokes need to meet the user specified edits.

To formulate edit propagation, a key is to define a similarity measure, or *affinity*, between pixels. While the exact definition varies among different propagation methods, pixel affinity is typically a function of both spatial locality and appearance. For example, in [An and Pellacini 2008], the affinity z_{ij} between two pixels i, j in an image is expressed as:

$$z_{ij} = exp(-\|\mathbf{f}_i - \mathbf{f}_j\|^2) \qquad (1)$$

where $\mathbf{f}_i = (\mathbf{c_i}/\sigma_c, \mathbf{p_i}/\sigma_p)$ is a feature vector at pixel i comprising of its appearance $\mathbf{c}_i$ (e.g., color in Lab space) and position $\mathbf{p}_i$ (e.g., in x and y coordinates) weighted by parameters σ_c, σ_p. For video, we can expand the feature vector to include the frame index t_i of pixel i, and introduce a new parameter σ_t so that $\mathbf{f}_i = (\mathbf{c_i}/\sigma_c, \mathbf{p_i}/\sigma_p, \mathbf{t_i}/\sigma_t)$. In the subsequent parts of our paper, we report the values of σ_p and σ_t normalized to image resolution and video length, respectively.

Affinity-based edit propagation can be formulated as an energy minimization problem. Given user specified edits (e.g., amount of change in color, contrast, etc.) at some pixels, the goal is to compute the edits at all pixels so as to minimize the difference in edits between pixels with high affinity on one hand, and the deviation to user-provided edits on the other hand. While the exact formulations of propagation minimization vary in literature [Lischinski et al. 2006; Pellacini and Lawrence 2007; An and Pellacini 2008], the minimization is typically performed on a per-pixel basis,

Figure 2: Adaptive clustering in affinity space. (a) An image with two user strokes. (b) Pixel clusters colored by their distances to the closest user-edited pixel in the affinity space. Note that clusters further away from the strokes, which are dimmer, have larger sizes.

which makes the computational cost prohibitive for large images or videos.

To reduce the complexity of solving propagation minimization, we replace individual pixels by clusters of pixels as variables in the minimization. Specifically, we group pixels into clusters in which the propagated edits can be approximated by simple (e.g., linear) functions. We then solve for the coefficients of these functions, instead of the edits at individual pixels, by re-formulating the pixel-based propagation energy. We observed that, using a hierarchical clustering technique in the affinity space, the propagated edits at all pixels can be well approximated using much fewer clusters (e.g., shown in Figure 2 (b)), hence resulting in dramatic improvement in the computational cost.

We will first present our clustering method for images and videos (Section 4). The clustering is guided by the general principles of affinity-based propagation, and therefore independent of and applicable to specific energy formulations used in the minimization. We will then demonstrate the use of clusters in minimizing a recently proposed, all-pixel-pairs propagation energy [An and Pellacini 2008] by tailoring the energy formulation to consider clusters instead of pixels (Section 5).

4 Affinity space clustering

To motivate our clustering method, we start with a different interpretation of the principles used in affinity-based edit propagation. Consider a higher-dimensional space where pixels in an image are represented as points in this space, so that two points representing pixels with a high affinity are located closed-by. We call this space an *affinity space*. According to the first propagation principle, the propagated edits in the image space would form a *smooth function* in the affinity space. This is because nearby points in the affinity space represent highly "similar" pixels and therefore should receive similar edits. Based on the second principle, the smooth edit function is constrained by user-given edits at a collection of points representing the stroke pixels.

To efficiently compute this edit function, we draw our inspiration from the approach of Agarwala [2007] for solving gradient domain composition problems. Like edit propagation, image composition also looks for a function (e.g., color increments at each pixel) that is smooth across neighboring pixels while satisfying constraints at certain locations (e.g., the composition seams). Agarwala showed that such function can be well approximated using linear pieces that are smaller near the constrained regions and larger elsewhere.

Likewise, we will approximate the edit function in the affinity space by linear pieces, each composted of a cluster of pixels, that are smaller near the stroke pixels and larger elsewhere. While a quadtree partitioning scheme is used in [Agarwala 2007] to obtain the
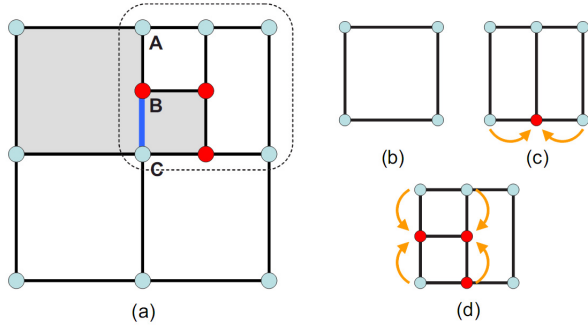
Figure 3: (a) A 2D k-d tree with T-junctions (red). (b-d) Illustration of a recursive algorithm for computing interpolated values at T-junctions in the bracketed part of (a).

clusters and to represent the linear function within each cluster, we extend the scheme to a k-d tree since the affinity space is high-dimensional.

Affinity space The definition of the affinity space depends on the choice of affinity measures. In our implementation, we adopt the affinity measure in Equation 1. As a result, the affinity space is defined simply by representing a pixel i using its feature vector f_i, since pixels with higher affinity have smaller Euclidean distances between their feature vectors. Note that the discussion in the rest of the section can be applied generally to any affinity space definitions.

K-d tree partitioning To construct the pixel clusters, we divide up the affinity space using a k-d tree that is more finely subdivided in regions closer to the user-edited points. We build the tree in a top-down fashion. Starting from a root cell, the tightest bounding cube of points representing all pixels in the image, we recursively split a cell to two child cells midway along an axis that is alternated at successive tree levels. The splitting stops if the cell contains no data point, or if the diagonal of the cell minus the distance from the cell center to the nearest user-edited point is smaller than a user-given threshold η. All pixels represented by points in a leaf cell of the k-d tree are then considered as a cluster. The stopping criteria ensures that the size of cells linearly increases with the distance from the user-edited points (see Figure 2 (b)), whereas the parameter η effectively controls the size of the smallest cell.

Building piece-wise linear functions The k-d tree partitions the affinity space into disjoint cells, in which the edit function can be approximated linearly. Specifically, the corners of the non-empty leaf cells will be used instead of the actual pixels in performing the propagation computation (see Section 5). Afterwards, the edits at individual pixels within a leaf cell are simply obtained by multi-linear interpolation from those at the cell corners.

Care must be taken to ensure the continuity of these linear functions across neighboring cells with different sizes. In the 2D example in Figure 3 (a), linear interpolation within each shaded cell using the computed edits at their four corners will not match along the highlighted edge if the value at corner B is not the average of those at A and C. Agarwala [2007] resolves the problem on a quad-tree by using a restricted quad-tree (i.e., no two cells that share an edge may differ in tree depth by more than one) and enforcing the value at a "T-junction" (e.g., corner B) to be the average of the values at the two ends of the edge (e.g., A and C). We generalize this idea to unrestricted k-d trees in arbitrary dimensions, and give a simple recursive algorithm for determining values at T-junctions that ensures the continuity of the resulting piece-wise linear function.

We explain the algorithm using the 2D k-d tree example in Figure 3 (a). Given a k-d tree in a d-dimensional space, we define a "T-junction" as a cell corner that is contained in fewer than 2^d leaf cells. If the corner lies on the boundary of the root cell (e.g., A in Figure 3 (a)), we change d to be the dimensionality of facet that the corner lies on (e.g., $d = 1$ for A as it is on an edge). These T-junctions (colored red in Figure 3 (a)) can be tagged in a single traversal of the k-d tree. To compute the enforced value at each T-junction, we perform a second top-down walk from the root cell as illustrated in Figure 3 (b-d). At a cell (e.g., the one in (b)), we consider the corners intersected by its splitting plane and the cell *edges* (e.g., the two points in the middle column of (c)). For each corner that is a T-junction, we assign it the average of the values at the two end corners of the underlying cell edge (indicated by arrows in (c)). The process is repeated in each of its child cells (as in (d)).

4.1 Analysis

Tree complexity As observed in Figure 2 (b), large areas of the image are grouped into common clusters, indicating that much fewer clusters are used than the number of actual pixels. The number of clusters depends on several factors including the number of user-edited pixels, the distribution of the pixels in the affinity space, and the depth of the k-d tree. In general, more clusters are constructed if the number of user-edited pixels increases or the pixels are more uniformly distributed in the affinity space. If both the image and the user-strokes are fixed, the cluster number grows with the depth of the k-d tree, which is effectively governed by the size of the smallest leaf cell (η) and the size of the root cell that represents the bounding box of all pixels. Using the affinity definition in Equation 1, the bounding box size is inversely proportional to the parameters $\sigma_c, \sigma_p, \sigma_t$. As a result, the number of clusters decreases as any one of the parameters $\eta, \sigma_c, \sigma_p, \sigma_t$ increases.

To more precisely quantify the system complexity, we consider the number of corners of non-empty leaf cells on the k-d tree, which are the actual variables that will be used in computing edit propagation. Figure 4 plots the cell corner number as a function of η and σ_p for the image and video example shown in Figure 6. Observe that, even at extremely small values of η (corresponding to high approximation accuracy) and σ_p (corresponding to highly localized editing effects), the number of cell corners is still much smaller than the number of pixels. Also note that the cell corner number is *not* strongly affected by the increase in the number of pixels in the data. Although the video data has 300 times more pixels than the image, the number of cell corners grows only by 2 times. This is because video data contains strong coherence between pixels, and hence the pixels tend to exhibit much less uniform distribution in the affinity space than those in an image.

Space and time complexity Both the tree-building and interpolation stages take $O(dn)$ time for an image or video with n pixels,
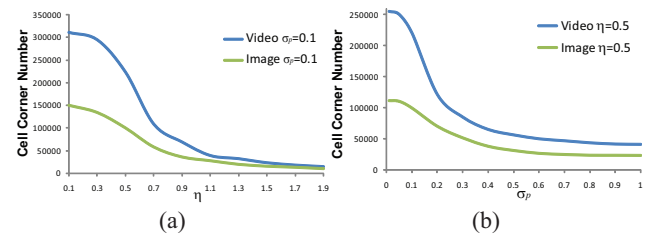

Figure 4: The number of leaf cell corners on the k-d tree as a function of minimum cell size η (a) and affinity parameter σ_p (b), for the bird video (containing 61M pixels) and the fog tree image (containing 0.2M pixels) in Figure 6.

where d is the average depth of leaf cells on the k-d tree for all pixels, because both involve identifying the leaf cell to which a pixel belongs. The storage for the whole tree is $O(c)$ where c is the number of cell corners of non-empty leaf cells.

5 Efficient edit propagation

Here we show how the cluster-based approximation is applied to edit propagation. A number of energy formulations for propagation have been proposed before [Lischinski et al. 2006; Pellacini and Lawrence 2007; An and Pellacini 2008]. Here we consider the recent formulation of [An and Pellacini 2008], which is designed to better support sparse edits to propagate over spatially discontinuous regions. Specifically, given user specified edits g_i with strength $w_i \in [0, 1]$ (0 indicating no user constraints) at each pixel, it computes the edits e_i at all pixels by minimizing the energy form:

$$\sum_{i,j} w_j z_{ij}(e_i - g_j)^2 + \lambda \sum_{i,j} z_{ij}(e_i - e_j)^2 \quad (2)$$

where the affinity z_{ij} is defined as in Equation 1 and i, j enumerates over all pixel pairs. The two minimization terms in the sum respectively express the two guiding principles we mentioned earlier, the satisfaction of user constraints and the similarity of edits in similar pixels. λ is set to $\sum_i w_i/n$ to make the relative contributions of the two terms similar, where n is the number of pixels. Propagation to spatially discontinuous regions is realized by minimizing over all pairs of pixels weighted by their affinity.

One way to utilize our k-d tree clustering in this formulation is to express the variables e_i in Equation 2 as linear combinations of cell corners in the tree, which would subsequently result in a linear system with much fewer variables to solve. However, constructing the coefficients in the linear system would still be rather time consuming due to the enumeration of all pair-wise terms.

Alternatively, we consider replacing the variables e_i, which are unknown edits at individual pixels, directly by new variables $\bar{e}_j$ that are edits at cell corners in the k-d tree. In this way, the enumeration in Equation 2 would be over all pairs of cell corners, which is drastically fewer than all pairs of pixels. In a sense, we are now solving the propagation problem on a different set of samples in the affinity space rather than the original pixels. Since the density of these samples adapts to the variation of the edit function (e.g., denser in regions closer to the user-edited points), solving the edit function directly on these samples would yield a reasonable approximation, which we have confirmed in our experiments.

Specifically, the modified formulation has the form:

$$\sum_{i,j} \bar{w}_j \mu_i \mu_j z_{ij}(\bar{e}_i - \bar{g}_j)^2 + \lambda \sum_{i,j} \mu_i \mu_j z_{ij}(\bar{e}_i - \bar{e}_j)^2 \quad (3)$$

where i, j now enumerates over all pairs of cell corners on the k-d tree. Here, $\bar{g}_i, \bar{w}_i$ are user edit and edit strength at cell corner i, and are computed as weighted sums of user edits and strengths at pixels in the neighboring cells. Specifically, $\bar{w}_i = \sum_k s_{ik} w_k / \sum_k s_{ik}$ and $\bar{g}_i = \sum_k s_{ik} g_k w_k / \sum_k s_{ik} w_k$ where s_{ik} is the multi-linear barycentric coordinate of pixel k in a neighboring cell of corner i with respect to corner i. Note that $\sum_k$ enumerates over all pixels in any adjacent cell of corner i. Finally, the term μ_i compensates for the multiplicity of pixels in the neighboring cells of corner i, computed as $\mu_i = \sum_k s_{ik}$.

Minimization and analysis The modified energy in Equation 3 is minimized in the same way as Equation 2 is minimized in [An and Pellacini 2008], which involves solving a linear system of equations with n variables, where n is the number of pixels. Using an efficient stochastic column sampling approach, the time and space

(a) original image and user strokes

(b) edited image

Figure 5: Editing a tree image of resolution 3720×1140. The top image indicates the original image and user input strokes. The bottom image is the edited result. The parameters used are $\sigma_c = 0.2$, $\sigma_p = 0.2$.

complexity of the solver in [An and Pellacini 2008] are respectively $O(m^2 n)$ and $O(mn)$, where m is the number of sampled columns. Minimizing our modified energy in Equation 3, which involves only c variables where c is the number of k-d tree cell corners, using the same procedure would yield a reduction on the order of n/c in both space and time. We have observed that this ratio n/c ranges from tens to thousands in our test data, and particularly large for video.

6 Comparisons and examples

We show a number of comparisons and examples to demonstrate the effectiveness of our cluster-based approximation. To be able to compare with the original formulation in [An and Pellacini 2008] on large data, we implemented an out-of-core version of their algorithm. Note that our approximated propagation is completely in-core. Our tests were performed on a PC with Intel I7 920 processor (2.66GHz, 4cores) and 4GB memory.

Comparison We first compare in Figure 6 our approximation with the propagation results obtained by the original formulation in [An and Pellacini 2008]. We experimented with increasing value of η, which is the only parameter used to control our approximation. Observe that higher values of η incur greater approximation errors, even at user-provided strokes, as pixels are approximated by corners of larger k-d tree cells (here the error percentage is the RMS difference in absolute pixel intensities). On the other hand, low values of η (e.g., ≤ 1.0) produce results that are visually indistinguishable from the original method of [An and Pellacini 2008], yet yielding a substantial reduction in time and storage cost, as reported in Table 1 for $\eta = 0.5$.

In this test, we used sampling column number $m = 100$ in solving the linear systems, which achieves a good sampling accuracy for both large ($\sigma_p = 1.0$) and small ($\sigma_p = 0.1$) values of σ_p. A significantly large m needs to be used for very small values of σ_p (e.g., less than 0.01), as the rank of the matrix in solving the linear system increases. However, we did not consider these values in our comparisons as the propagation formulation in [An and Pellacini 2008] is designed primarily for achieving global editing effects where sparse edits are expected to propagate thorough spatially discontinuous regions.

More examples With the greatly improved efficiency, our method can be applied for interactive editing of high-resolution images (Figure 5) and long (hundreds of frames) video sequences (Figure 1 and Figure 7). These examples demonstrate color, saturation
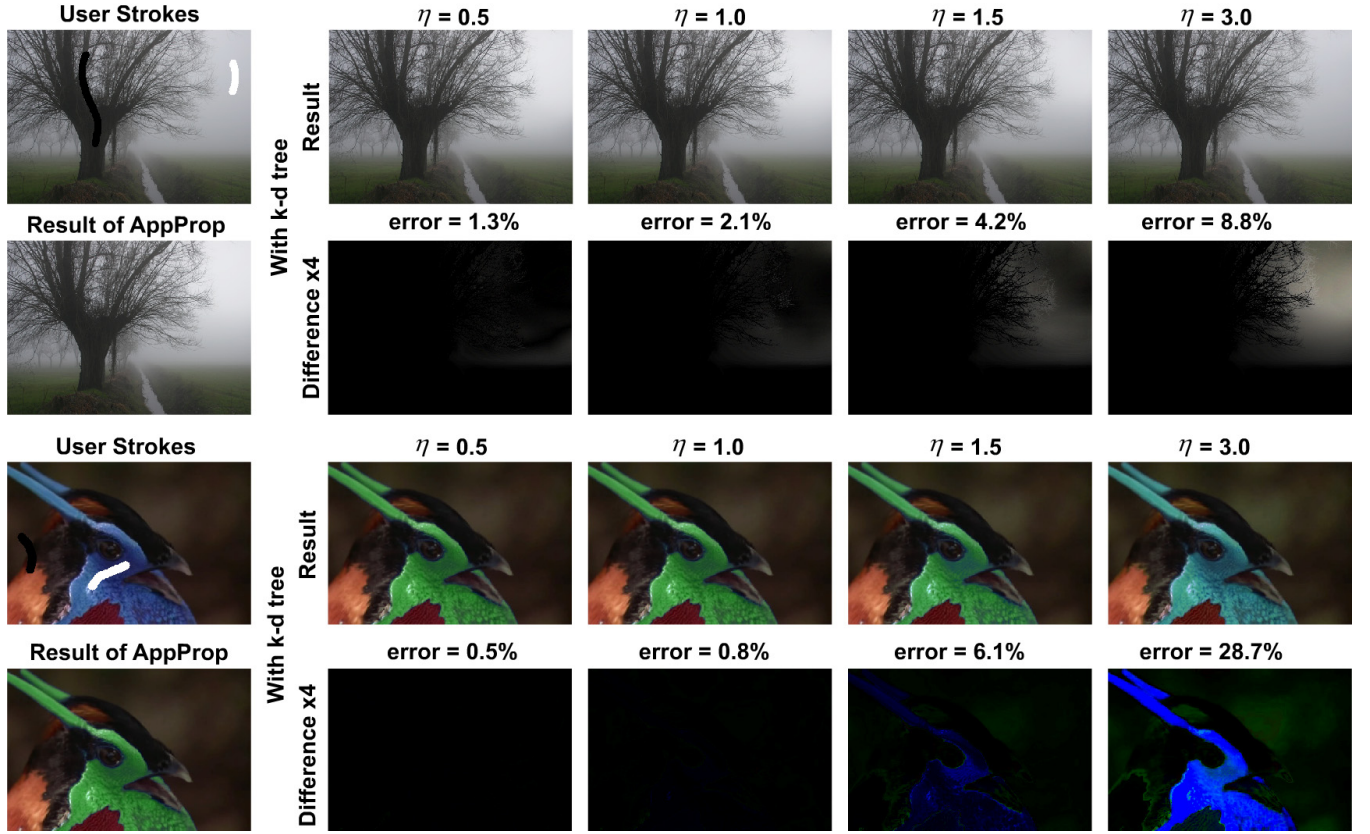
Figure 6: Comparison between our approximation and AppProp [An and Pellacini 2008]. We show edited results of our approximation and the difference images(multiplied by 4 times) to the results of AppProp, using different η on the fog tree image (top 2 rows) and the bird video (bottom 2 rows). The parameters used are $\sigma_c = 0.2$ and $\sigma_p = 0.1$ (the fog tree image), and $\sigma_c = 0.2$, $\sigma_p = 1.0$ and $\sigma_t = 1.0$ (the bird video).
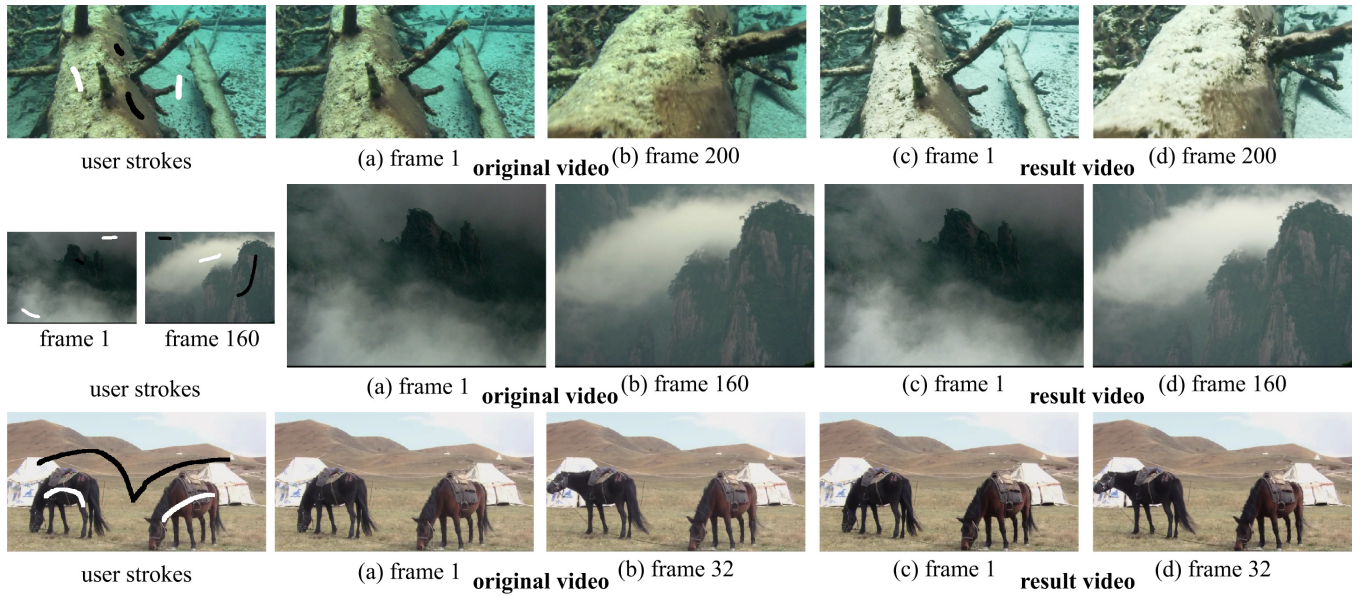

Figure 7: Adjusting the color of the lakebed video (top row, with parameters $\sigma_c = 0.2$, $\sigma_p = 1.0$ and $\sigma_t = 1.0$), adjusting the brightness of the fog video (middle row, with parameters $\sigma_c = 0.2$, $\sigma_p = 1.0$ and $\sigma_t = 1.0$) and adjusting the tone of the horse video (bottom row, with parameters $\sigma_c = 0.3$, $\sigma_p = 0.4$ and $\sigma_t = 1.0$), showing user strokes, two selected frames before (a,b) and after (c,d) editing.

Data	tree	bird	horse	fog	lakebed
Type.	image	video	video	video	video
Res.	4M	61M	36M	47M	55M
with k-d tree					
Corner No.	48k	30k	28k	15k	10k
Time.Build.	0.3s	2s	1s	2s	2s
Time.Prop.	0.4s	0.3s	0.3s	0.1s	0.1s
Time.Interp.	0.4s	6s	4s	5s	5s
Time.Total.	1s	8s	5s	7s	7s
Memory	36M	22M	21M	11M	8M
RMS Error	1.9%	0.5%	0.8%	1.1%	1.2%
AppProp (without k-d tree)					
Time	29s	48min	29min	37min	44min
Memory	1.6G	23G	14G	18G	21G

Table 1: Performance comparison of an out-of-core implementation of [An and Pellacini 2008] (without k-d tree) and our in-core approximation (with k-d tree). For our approximation, it shows the number of cell corners, execution time for the three stages of the algorithm (e.g., tree-building, edit propagation using cell corners, interpolation to pixels), total execution time, memory cost, and the RMS error.

and tone adjustment to user desired regions on images and videos containing complex textures and fog scenes. In all our examples, we use sampling column number $m = 100$ and minimum cell size $\eta = 0.5$. Note that videos are treated in the same way as images in our method, except for the additional dimension (frames) and the additional parameter (σ_t) in the affinity definition (Equation 1). No tracking or explicit correspondence finding is used in these examples. The computations in these examples range from 1 second on images and 8 seconds on videos, and never take more than 150MB of memory. The detailed running time and memory consumption are reported in Table 1. Observe that our cluster-based approximation yields hundreds of times improvement to the out-of-core implementation of [An and Pellacini 2008]. Note that the interpolation time in video editing can be substantially improved for fast *previewing*, where the user desires to see the propagated result just for a single frame of the video, and only the edits of the pixels on that frame need to be interpolated from those on the k-d tree cell corners.

7 Conclusion

We present a novel paradigm for solving energy minimization problems in affinity-based edit propagation. We reduce the size of the problem by considering a set of sparse samples in the affinity space instead of individual pixels in the input image or video. The samples are placed at corners of a k-d tree that is adaptively subdivided in the affinity space to capture the variation in the desired edit function. With our approximation scheme, we have observed that edit propagation takes substantially less time and memory without sacrificing visual fidelity. Our new paradigm for solving edit propagation dramatically extends the practical capabilities of existing pixel-by-pixel propagation approaches to handle high-resolution images and long video sequences directly in-core at an interactive speed.

There are a number of limitations in our current method that we wish to address in future works. For example, the complexity of our method increases with more user strokes, and hence is not suited for performing "dense" edits. Also, the increase of complexity in achieving more local editing effects can be undesirable for fine-tuning the results. Other research venues include exploring ways to edit streaming contents, and to investigate other areas of image/video processing where this clustering technique would be useful.

References

ADAMS, A., GELFAND, N., DOLSON, J., AND LEVOY, M. 2009. Gaussian kd-trees for fast high-dimensional filtering. *ACM Trans. Graph. 28*, 3, 21.

AGARWALA, A. 2007. Efficient gradient-domain compositing using quadtrees. *ACM Trans. Graph. 26*, 3, 94.

AN, X., AND PELLACINI, F. 2008. Appprop: all-pairs appearance-space edit propagation. *ACM Trans. Graph. 27*, 3, 40.

CHEN, J., SYLVAIN, P., AND FRÉDO, D. 2007. Real-time edge-aware image processing with the bilateral grid. *ACM Trans. Graph. 26*, 3, 103.

CHESLACK-POSTAVA, E., WANG, R., AKERLUND, O., AND PELLACINI, F. 2008. Fast, realistic lighting and material design using nonlinear cut approximation. *ACM Trans. Graph. 27*, 5, 128.

FATTAL, R., CARROLL, R., AND AGRAWALA, M. 2009. Edge-based image coarsening. *to appear in ACM Trans. Graph.*.

FATTAL, R. 2009. Edge-avoiding wavelets and their applications. *ACM Trans. Graph. 28*, 3, 22.

HANRAHAN, P., SALZMAN, D., AND AUPPERLE, L. 1991. A rapid hierarchical radiosity algorithm. *SIGGRAPH Comput. Graph. 25*, 4, 197–206.

KOPF, J., COHEN, M. F., LISCHINSKI, D., AND UYTTENDAELE, M. 2007. Joint bilateral upsampling. *ACM Trans. Graph. 26*, 3, 96.

LEVIN, A., LISCHINSKI, D., AND WEISS, Y. 2004. Colorization using optimization. *ACM Trans. Graph. 23*, 3, 689–694.

LI, Y., H., A. E., AND ASEEM, A. 2008. Scribbleboost: Adding classification to edge-aware interpolation of local image and video adjustments. *Computer Graphics Forum 27*, 4 (June), 1255–1264.

LISCHINSKI, D., FARBMAN, Z., UYTTENDAELE, M., AND SZELISKI, R. 2006. Interactive local adjustment of tonal values. *ACM Trans. Graph. 25*, 3, 646–653.

PELLACINI, F., AND LAWRENCE, J. 2007. Appwand: editing measured materials using appearance-driven optimization. *ACM Trans. Graph. 26*, 3, 54.

SHI, J., AND MALIK, J. 2000. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 22*, 8, 888–905.

WALTER, B., FERNANDEZ, S., ARBREE, A., BALA, K., DONIKIAN, M., AND GREENBERG, D. P. 2005. Lightcuts: a scalable approach to illumination. *ACM Trans. Graph. 24*, 3, 1098–1107.

WALTER, B., ARBREE, A., BALA, K., AND GREENBERG, D. P. 2006. Multidimensional lightcuts. *ACM Trans. Graph. 25*, 3, 1081–1088.

YATZIV, L., AND SAPIRO, G. 2006. Fast image and video colorization using chrominance blending. *IEEE Transactions on Image Processing 15*, 5, 1120–1129.