Efficient Antialiased Edit Propagation for Images and Videos

Li-Qian Ma[a], Kun Xu[a,*]

[a]*TNList, Department of Computer Science and Technology, Tsinghua University, Beijing*

Abstract

Edit propagation on images/videos has become more and more popular in recent years due to simple and intuitive interaction. It propagates sparse user edits to the whole data following the policy that nearby regions with similar appearances receive similar edits. While it gives a friendly editing mode, it often produces aliasing artifacts on edge pixels. In this paper, we present a simple algorithm to resolve this artifact for edit propagation. The key in our method is a new representation called Antialias Map, in which we represent each antialiased edge pixel by a linear interpolation of neighboring pixels around the edge, and instead of considering the original edge pixels in solving edit propagation, we consider those neighboring pixels. We demonstrate that our work is effective in preserving antialiased edges for edit propagation and could be easily integrated with existing edit propagation methods such as [1, 2].

Keywords: Antialiasing Recovery, Edit Propagation, Antialias Map

1. Introduction

With the development of digital image/video cameras and online image/video sharing services (e.g. flickr, youtube), it is much easier for people to access images/videos than before. The desire to edit the appearance of image/video, such as color, brightness, tonal values, arises. One way to edit the appearance of images is to first select some regions of interest, and then apply a desired edit operation to those regions. While this is a common solution in commercial softwares such as Photoshop, selecting those regions of interest, is still a time consuming task, especially for images with complex textures. Another way is to use edit propagation methods [1, 2, 3]. In these methods, users only need to specify sparse strokes indicating specific edits (as shown in Figure 1 (a)), and those edits would be automatically propagate to the whole data following the policy that nearby regions with similar colors receive similar edits.

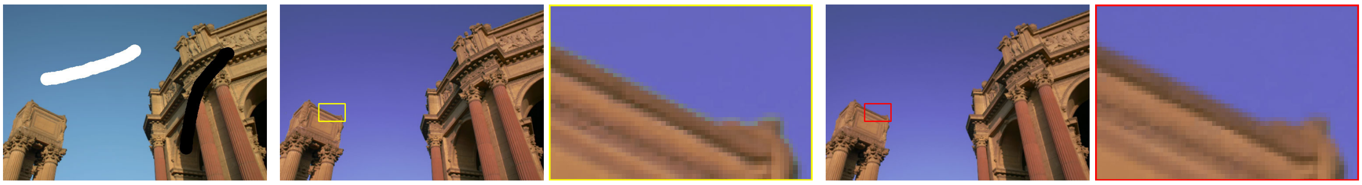
While edit propagation methods provide a much simpler and more convenient way for editing images/videos, it often suffers from a visible aliasing artifact. As illustrated in Figure 1, in this example, users draw a white stroke on the sky and a black one on the building, indicating an edit operation that changes color and another edit operation that keeps original color, respectively. Figure 1 (b) gives the result generated by a state-of-the-art edit propagation work [1], while it achieves the goal in most parts of the image, however, as shown in the enlarged image in (b), along the boundary of the building, we see an undesired, clear edge.

It's not surprising that edit propagation methods would produce such aliasing artifacts. This is simply because edit propagation is a per-pixel algorithm and would fail on antialiased pixels. Take Figure 1 as example, in the original image (in Figure 1 (a)), due to its antialiasing nature, the edge pixels exhibit neither the color of sky nor the color of the building, but a kind of blending between the colors of sky and the building. However, under the policy of edit propagation, those antialiased edge pixels are neither similar to the sky pixels nor to the building pixels due to color differences, this makes appearance of those edge pixels unchanged after edit propagation, leading to antialiased edges damaged, as shown in Figure 1 (b). The existence of such artifacts, has largely reduced the fidelity of results and practicability of edit propagation.

To address this issue, in this paper we introduce a novel, efficient framework to eliminate those aliasing artifacts in edit propagation. Our work is inspired by a recent work on antialiasing recovery [4], which aims at restoring antialiased edges for a range of image filters. Similar to [4], we assume that for antialiased edges in images, the value of each pixel could be seen as a linear interpolation from some nearby pixels. Based on this assumption, we introduce a novel representation, the Antialias Map, which stores the blending weights and relative positions of nearby interpolating pixels for each edge pixel. While previous works [1, 2, 3] directly consider edge pixels in solving edit propagation, we replace each edge pixel by its interpolating pixels and use those interpolating pixels in edit propagation instead. In turn, the edits of each edge pixel is obtained by an interpolation from those interpolating pixels. As shown in Figure 1 (c), our method successfully preserves the smooth edge around the boundary of the building after edit propagation. Furthermore, our method is independent of a specific edit propagation algorithm and could be integrated into any existing edit propagation methods such as [1, 2, 3]. The results demonstrate that our method effectively preserves the antialiased smooth edges without incurring large performance overhead.

The rest of the paper is organized as follows: we will first review some important related works in edit propagation and

*Corresponding author



(a) original image (b) result of Xu et al. (c) our result

Figure 1: *An example of edit propagation. (a) shows the original image and user strokes. (b) and (c) show the propagation results using the method by [1] and our method, respectively. Alias artifacts are visible in (b) along the boundary of the building. Our method successfully eliminate these artifacts, as shown in (c).*

antialiasing recovery, respectively, in Section 2; the Antialias Map will be introduced in Section 3; the framework and algorithm details for edit propagation will be explained in Section 4; after that, results and comparisons will be given in Section 5 and conclusions will be made in Section 6.

2. Related Works

In this section we will review some important prior works in edit propagation and antialiasing recovery, respectively.

2.1. Image/Video Editing

Image/video editing is an increasingly hot topic in computer graphics in recent years. It could be generally divided into two groups: structural editing [5, 6, 7, 8, 9, 10] and appearance editing. Appearance editing includes tone editing [11, 12, 13, 14, 15, 16], colorization [17, 18, 19, 20], dehazing [21, 22, 23, 24, 25], and edge-aware editing [26, 27, 28, 29, 30, 31], etc.. Recently, edit propagation methods [1, 2, 3, 32] allow a simpler interaction mode for appearance editing. In these methods, users specify edits in some sparse locations on images/videos, and those edits are automatically propagated to the whole data satisfying the policy that nearby pixels having similar appearances receive similar edits. Usually, edit propagation methods define affinities between pairs of pixels according to their appearance/position distances, and different optimization schemes are utilized to satisfy the policy. In particular, Pellacini et al. [32] approximate pixel relations using a sparse graph and reduce edit propagation problem to solving a sparse linear system. An and Pellacini [3] introduced a more robust algorithm, which considers all-pairs affinities between pixels, and approximates the huge affinity matrix using a low rank approximation. To accelerate edit propagation, Xu et al. [1] uses a k-d tree to organize pixels into hierarchical clusters in a high dimensional space, instead of propagating on individual pixels, they propagate on clusters which largely reduced time and memory cost. Xiao et al. [33] employs a similar hierarchical structure for acceleration. Li et al. [2] further speeds up edit propagation by formulating it as a function interpolation problem. Bie et al. [34] accelerate edit propagation using static clustering and efficient sampling scheme. Besides images and videos, edit propagation could be also used to edit spatially varying bidirectional reflectance distribution functions obtained by [35, 36, 37, 38] and bidirectional texture functions [39, 40]. Recently, Farbman et al. [41] proposes to use diffusion distance, instead of Euclidean distance, to define affinities between pixels, which better account for the global distribution of pixels.

2.2. Antialiasing Recovery

In computer graphics, many techniques have been proposed to render antialiased images [42, 43], antialiased textures [44, 45] and antialiased shadows [46, 47, 48]. However, only a few works focus on recovering smooth, antialiased edges from aliased 2D images. Some exceptional works include Principle Component Analysis (PCA) [49, 50] and morphological antialiasing [51]. In particular, morphological antialiasing aims at reducing aliasing artifacts for rendered images entirely using image based methods. It looks for certain patterns of discontinue geometry and replace them using smooth edges estimated by an antialiased edge model. Image vectorization techniques [52, 53, 54] convert a bitmap image to a vectorized image, which could also be used to antialias certain types of images. Recently, Yang et al. [4] introduced a method for recovering antialiased edges destroyed by a range of non-linear image filters. In this work, an analytic edge model is estimated using the original image, and is applied to the filtered image to remove aliasing artifacts. It works well for many non-linear image filters such as intensity thresholding, tone mapping, color to gray and so on, however, since it requires perfect pixel correspondence between the original and filtered images, it cannot handle filters like Gaussian blurring. Besides, it's not clear how to extend this method to edit propagation.

Compared to the conference paper [55], We have extended our framework to handle interpolation based edit propagation. This is a significant new contribution compared to [55], since we have demonstrated the proposed Antialias Map is not limited to optimization based edit propagation, however, it could also be used for interpolation based edit propagation. This demonstrates that the proposed Antialias Map is independent with specific edit propagation methods and could be potentially combined with any edit propagation methods.

3. Antialias Map

As mentioned before, since antialiased edges in images are often smooth, we assume the value of an edge pixel could be approximated by a linear interpolation of some nearby pixels. We present Antialias Map to store those edge pixels. Besides, in Antialias Map, for each edge pixel ,we also store the information of its neighboring interpolating pixels, including both interpolating weights and relative positions. For videos, we store an Antialias Map for every frame. Since our work is built upon the antialiasing recovery work of [4], to make our paper self-contained, before introducing the details of Antialias Map, we will first explain some necessary backgrounds in [4] in Section 3.1.

3.1. Antialiasing Recovery

Images often have smooth, antialiased edges. However, these desired properties will be destroyed by a range of nonlinear image filters, such as intensity thresholding, tone mapping, etc.. After applying those image filters, smooth boundaries become zigzag like. Yang et al. [4] proposed a technique to remove these aliasing artifacts in filtered images. Their method proceeds in several steps:

Edge model. For each pixel i in the original image, they choose the two extremum colors c_j and c_k (j,k are corresponding pixels) in the principle direction of color space from the neighboring 8 pixels (in 3×3 size neighborhood). The principle direction is determined using an Expectation Maximazation (EM) scheme. Using extremum colors to reconstruct the color c_i of pixel i, the interpolation weights α_{ij},α_{ik} could be determined by minimizing:

$$d_i = \left\|(\alpha_{ij}c_j + \alpha_{ik}c_k) - c_i\right\| \tag{1}$$

where it satisfies $\alpha_{ij} + \alpha_{ik} = 1$.

Probability of lying on edges. After that, they estimate the probability of each pixel that it lies on an edge. For each pixel i, They define an edge strength e_i, which is the product of the Sobel edge detector at both the original image and the filtered image. The probability value of a pixel lying on an edge is defined as:

$$\beta_i = G(d_i, \sigma_d)(1 - G(e_i, \sigma_e)) \tag{2}$$

where $G(d, \sigma)$ is a 1D Gaussian defined as $exp(-d^2/\sigma^2)$, d_i is the residual distance defined in Equation 1, σ_d and σ_e are two controllable parameters. β_i is set as zero if $e_i > 3\sigma_e$.

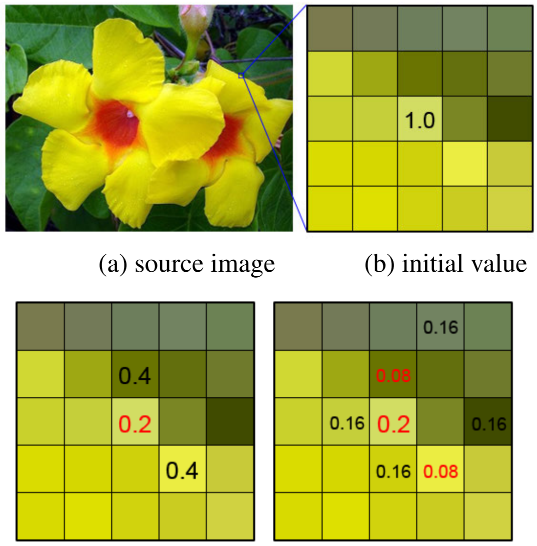
Recovery the filtered image. Denote f_i is the color value of pixel i on the filtered image. The recovered color value r_i could be obtained by solving the linear system below:

$$r_i = \beta_i(\alpha_{ij}r_j + \alpha_{ik}r_k) + (1 - \beta_i)f_i \tag{3}$$

This is a large sparse linear system and could be solved efficiently by a few iterations using the Jacobi method.

3.2. Compute Antialias Map

As discussed in Section 3.1, in [4], the value of each antialiased edge pixel is approximated by a blending of 2 nearby pixels in the 3×3 neighborhood. Results are progressively refined by iterations of Equation 3. Instead of using a 3×3 neighborhood, Antialias Map approximates the value of each pixel by a blending of pixels from a larger neighborhood:



(a) source image (b) initial value

(c) 1 iteration (d) 2 iterations

Figure 2: *Antialias Map construction. (a) is the source image; (b),(c) and (d) give the Antialias Map of a certain pixel after 0,1,2 iterations, respectively. Divisible pixels are colored black, while indivisible pixels are colored red.*

$$c_i \approx \sum_j w_{ij}c_j \tag{4}$$

where j is the *interpolating pixel* in the neighborhood of i, and w_{ij} is the *interpolating weight* from pixel i to j, and satisfies $\sum_j w_{ij} = 1$. Note that w_{ij} does not necessarily equal to w_{ji}. Also note that Equation 4 is not an optimization target, and the interpolating weights are not solved from Equation 4. Instead, the interpolating weights are computed through an iteractive approach, which will be explained in detail below.

Antialias Map has two advantages over the edge model proposed in [4]. First, since it uses a larger neighborhood to approximate an antialiased pixel, it leads to a more accurate approximation; Secondly, the Antialias Map only depends on the structure of original image itself, it could be computed and stored before edit propagation, so it avoids the cost of iterations at run-time edit propagation stage. Antialias Map stores all interpolating weights w_{ij}, and it is sparse since it only considers those edge pixels (e.g. whose edge strength β_i is non-zero) and it only stores non-zero weights. Specifically, we store a set of triples $(\Delta x_{ij}, \Delta y_{ij}, w_{ij})$ for each edge pixel i. Here j is its interpolating pixel, Δx_{ij},Δy_{ij} and w_{ij} are the x,y position offset and interpolating weight from i to j, respectively. In the following parts, we will explain how to compute the Antialias Map in detail.

Initialization. In this step, we first use [4] to obtain the two extremum neighbors j, k, the blending factors α_{ij},α_{ik} and the edge probability β_i for each pixel i. We have already explained how to compute those values in Section 3.1. Care must be taken when computing the edge probability β_i. In [4], it defines edge strength of each pixel as the product of Sobel edge detector on

Initialization
For all pixels i
  Compute the blending factors α_{ij},α_{ik},
    and the edge probability β_i.
End For

Computation
Step 1: Antialias Map $S_i = \{\{0,0,1\}\}$
Step 2:
for each triple $\{\Delta x_{ij},\, \Delta y_{ij},\, w_{ij}\}$ in S_i
  if the pixel j is divisible and $\beta_j w_{ij} > \sigma_a$
    fetch blending factors α_{jk_1},α_{jk_2} and edge probability β_j;
    update the weight of pixel j to $(1-\beta_j)w_{ij}$;
    mark pixel j as indivisible;
    add pixel k_1 and k_2 to Antialias Map S_i, with weights
    $w_{ik_1} = \alpha_{jk_1}\beta_j w_{ij}$,$w_{ik_2} = \alpha_{jk_2}\beta_j w_{ij}$,
    mark these two pixels as divisible.
  end if
end for
if iteration number reaches N
  End.
else
  go back to Step 2.
end if

Table 1: *Pseudocode for Antialias Map Construction.*

both original and filtered images, which means it requires to obtain the aliased filtered image before antialiasing recovery. We observe that in edit propagation, the appearances are changed smoothly, so that the propagated result images have roughly the same structure as the original images. To avoid the cost to generate an aliased edit propagation result, we make a modification, instead, we define the edge strength as the Sobel edge detector only on the original image. Once the edge strength is computed, we use Equation 2 to compute edge probability β_i. Note that only the pixels with non-zero β_i are considered as antialiased edge pixels and stored in Antialias Map. The pixels with zero value of β_i are considered as non-edge pixels.

Constructions. Similar to [4], we construct Antialias Map by a few iterations. However, they obtain the final antialiased results through iterations, but we obtain Antilias Map through iterations, which could be precomputed and stored before edit propagation. For each antialiased edge pixel i, the Antialias Map starts with a set containing only one triple:

$$S_i = \{\{0,0,1\}\} \tag{5}$$

This means that the value of the pixel i could be seen as the value of itself multipled by weight 1.0, which is definitely true. We also illustrate this iteration process in Figure 2. As shown in Figure 2 (b), now the Antialias Map only contains itself with weight 1.0. And this pixel is marked as divisible, which is painted using black color in Figure 2.

At each iteration, we expand each divisible pixel (e.g. j) into 3 pixels. These 3 pixels are the two neighboring extremum pixels (e.g., k_1 and k_2) and itself (e.g. j), whose corresponding weights are defined in Equation 3. Specially, the weight of j is replaced by $(1-\beta_j)w_{ij}$ and j is marked as indivisible; the two newly added extremum pixels are marked as divisible, and their weights are set as $w_{ik_1} = \alpha_{jk_1}\beta_j w_{ij}$ and $w_{ik_2} = \alpha_{jk_2}\beta_j w_{ij}$, respectively. At the next iterations, we recursively find the di-

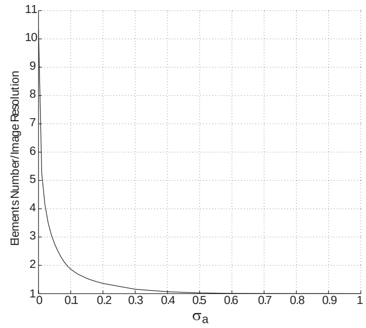


Figure 3: *the size of Antialias Map to the size of image as a function of threshold σ_a. This curve is generated from a 240K photographed image (the image in Figure 1) and using maximum iteration number $N = 4$.*

visible pixels and expand them to new pixels.

Let's also take Figure 2 as example and explain this process in detail. For simplicity, we assume that for all the pixels, the edge probability β is 0.8 and the blending factor α is 0.5. After the first iteration, the center pixel is expanded to 3 pixels, so that the Antialias Map grows to contain 3 triples (as shown in Figure 2 (c)):

$$S_i = \{\{0,0,0.2\},\{0,-1,0.4\},\{1,1,0.4\}\} \tag{6}$$

After the second iterations, similarly, the newly added 2 pixels in first iteration are both expanded to 3 pixels, so that the Antialias Map grows to contain 7 triples (as shown in Figure 2 (d)):

$$\begin{aligned} S_i = \{&\{0,0,0.2\},\{0,-1,0.08\},\{1,1,0.08\},\\ &\{-1,0,0.16\},\{1,-2,0.16\},\{0,1,0.16\},\{2,0,0.16\}\} \end{aligned} \tag{7}$$

Notice that at all iterations, the sum of weights equals to one. From an algebraic aspect, Antialias Map could also be treated as expanding Equation 3 to multiple variables. The triples in Antialias Map will extend to $(2N+1)\times(2N+1)$ neighborhood after n iterations.

Stop Criterion. The size of the Antialias Map grows as we iterate. We define 2 criterions to stop the recursive iteration:

- When iteration number reaches a predefined number N;
- When the result product(product of the interpolation weight of a divisible pixel w_{ij} and its edge probability β_j) is smaller than a predefined threshold σ_a.

The pseudocode of Antialias Map construction is given in Table 1. We have also tested how two parameters influence the performance of our algorithm. Figure 3 illustrates the size of Antialias Map to the size of image as a function of weight threshold σ_a. Setting $\sigma_a = 0$ means the iteration stops only when it reaches the largest iteration number N, while setting $\sigma_a = 1$ means no iteration. As shown in Figure 3, when increasing σ_a from 0 to 1, the size of Antialias Map decreases rapidly.

4. Improved Framework of Edit Propagation

In this section we will discuss how to use Antialias Map in the pipeline of edit propagation to remove the aliasing artifacts. In edit propagation, users specify edits in some sparse locations on images/videos, and those edits are automatically propagated to the whole data satisfying the policy that nearby pixels having similar appearances receive similar edits. Usually, they define a feature vector for each pixel, usually a 5-dimensional vector , which combines color (e.g. 3D), pixel position (e.g. 2D). For videos, another dimensional is added to account for time. The affinity between every two pixels are defined by the Euclidean distance between their feature vectors, which is then used to guide the propagation. Commonly, edit propagation methods could be divided into two groups, depending on which scheme is used to formulate the problem: optimization based [1, 3] and interpolation based [2]. We show that Antialias Map could be used in both groups for antialias recovery.

4.1. Optimization based Edit Propagation

Backgrounds. As mentioned above, the affinity value between two pixels i,j is usually defined as:

$$z_{ij} = exp\left(-(\mathbf{f}_i - \mathbf{f}_j)^2\right) \tag{8}$$

where $\mathbf{f}_i$ is the feature vector of pixel i, which is defined as a 5D vector for images:

$$\mathbf{f}_i = (c_i/\sigma_c, p_i/\sigma_p) \tag{9}$$

where c_i,p_i is the color in LAB color space and the pixel position of pixel i, respectively. σ_c and σ_p are two parameters to control the relative propagating distance.

In [3], edit propagation is formulated as an optimization problem. Solving propagated edits e is deduced to minimize the energy function below:

$$\sum_{i,j} b_j z_{ij}(e_i - g_j)^2 + \lambda \sum_{i,j} z_{ij}(e_i - e_j)^2 \tag{10}$$

where i,j enumerates all pixels; b_j is 1 when pixel j is covered by stroke and is 0 elsewhere; g_j is the user specified edit at pixel j; e_i is the propagated edit at pixel i that we want to solve. The first term accounts for how it satisfies user input while the second term accounts for the edit propagation policy that similar pixels receive similar edits. λ is used to control the relative weight between the two terms and is usually set to $\sum_j b_j / \sum_j 1$ to make the two terms have roughly the same contributions.

Since the energy function in Equation 10 is quadratic, minimizing it is equivalent to solving a linear system defined by a large affinity matrix. Therefore, they used low rank column sampling to approximate the affinity matrix and further proposed an approximated algorithm to fast find a solution. To accelerate edit propagation and extend it to videos, Xu et al. [1] proposed to use k-d tree to organize pixels into hierarchical clusters. Instead of propagating on pixels, they propagate on clusters, whose number is much smaller than the number of pixels, thus acceleration is achieved. Finally, edits of individual pixels are obtained by multi-linear interpolation from clusters. They also adopted an optimization based method to solve for edit propagation.

Modified Formulation. As illustrated in the teaser image, traditional edit propagation produces artifacts on object boundaries. This artifact could be easily explained. Assume a very simple image composed of 2 region, one red region and another blue region. The edge pixels along the boundary of the two regions would appear yellow due to antialiasing. Suppose user specifies some edits on the red region, it is also desired to propagate the edits to the edge pixels with some weight according to antialiasing opacity. However, since the edge pixels appearance yellow, it exhibits a large difference to pixels in the red region, hence would not receive any propagated edits.

To address this issue, we use Antialias Map, in which, the yellow edge pixels would be represented by a linear blending of some red and blue neighboring pixels. Instead of propagating to the edge pixels, we propagate to the neighboring interpolating pixels, and obtain the edit of edge pixel by blending the edits from the interpolating pixels. Mathematically, we modify the formulation in Equation 10 to:

$$\sum_{i,j} b_j \gamma_i \gamma_j z_{ij}(e'_i - g'_j)^2 + \lambda \sum_{i,j} \gamma_i \gamma_j z_{ij}(e'_i - e'_j)^2 \tag{11}$$

where i,j enumerates all interpolating pixels; γ_i considers the multiplicity of pixel i serving as interpolating pixels, which is defined as $\gamma_i = \sum_k w_{ki}$; g'_j is defined as $g'_j = \sum_k w_{kj} g_j / \sum_k w_{kj}$.

The modified energy function has the same form as the original energy function in Equation 10, so that it could be solved in the same way using either low rank column sampling [3] or k-d tree clustering [1].

Interpolation. After solving for the edits e' on the interpolating pixels in Equation 11, it is easy to obtain the edits on the edge pixels through interpolation:

$$e_i = \sum_j w_{ij} e'_j \tag{12}$$

4.2. Interpolation based Edit Propagation

Backgrounds. While most works adopt an optimization based method to solve edit propagation, Li et al. [2] proposed a different approach. They observe that the edits span in the high dimensional feature space form a smooth function, which could be approximated well by function interpolations. Therefore, they use sum of RBFs (Radial Basis Function) to approximate edits:

$$e_i \approx \sum_m a_m G(\|\mathbf{f}_i - \mathbf{f}_m\|) \tag{13}$$

where m iterates over all RBFs; G is RBF Gaussian function; a_m,$\mathbf{f}_m$ is the m-th RBF coefficient and center, respectively. The centers of RBFs are randomly selected from the pixels covered by user stroke. The coefficients of RBFs are solved by minimizing the sum of differences on user specified edits:

$$\sum_j (g_j - \sum_m a_m G(\|\mathbf{f}_j - \mathbf{f}_m\|))^2 \tag{14}$$

where j iterates over all pixels covered by user strokes. To restrict the coefficients to be non-negative, they use a non-negative least square solver.

Modified Formulation. The above formulation would also produce aliasing artifacts on object boundaries. To remove the artifacts using Antialias Map, similarly, we build the smooth function over the interpolating pixels, instead of the original pixels. Equation 14 is modified to:

$$\sum_j \gamma_j (g'_j - \sum_m a_m G(\|\mathbf{f}_j - \mathbf{f}_m\|))^2 \qquad (15)$$

where j iterates over all interpolating pixels that have contributions to user stroke pixels; γ_j considers the multiplicity of pixel j serving as interpolating pixels, which is defined as $\gamma_j = \sum_k w_{kj}$; g'_j is defined as $g'_j = \sum_k w_{kj} g_j / \sum_k w_{kj}$, where k is iterating over user stroke pixels.

After solving for the RBF coefficients, we use Equation 13 to obtain the edits on interpolating pixels. Lastly, we use Equation 12 to obtain the edits on the edge pixels.

5. Comparisons and Results

5.1. Comparisons

Comparison of weight threshold σ_a. In Figure 4, we have compared edit propagation results generated by Xu et al. [1] and by our method with different weight threshold σ_a. From the results, we can see artifacts using the method by Xu et al, where the pixels along the boundary of the toy undesirably appear green. Using a large value of σ_a (e.g. $\sigma_a = 0.8, 0.4$) still produce these artifacts. But using a relatively small value of σ_a (e.g. $\sigma_a = 0.1, 0.0$) fully removes the artifacts.

Comparison of maximum iteration number N. In Figure 5, we have compared edit propagation results generated by Xu et al. [1] and by our method with different maximum iteration number N. From the results, we can see that using a relatively large value of N (e.g. $N = 4, 8$) could produce smooth transitions along boundaries.

5.2. Results

All these results and performance are obtained using a consumer level PC with a 3.0GHz Intel Core2Duo CPU and 4GB RAM. As demonstrated in the comparisons, setting $\sigma_a = 0.1$ and $N = 4$ already leads to very good results. So in our implementation, we fix $\sigma_a = 0.1$ and $N = 4$. These two parameters could still be adjusted for better performance or accuracy. In our experiment, for a single image, the total size of Antialias Map (e.g. the total number of triples) is usually about $1.5 - 2.0$ times of the image resolution. So that it only needs small extra space to store the Antialias Map.

In Figure 6, we give 2 image results generated by the k-d tree approach [1] and by our method. In Figure 7, we give 2 image results generated by the RBF interpolation approach [2] and by our method. In Figure 8, we give 1 image result generated by AppProp [3] and by our method. In Figure 9, we compare a video example using the k-d tree approach [2] and using our method, respectively. In all these examples, after applying our methods, the aliasing artifacts along the object boundaries are successfully removed. The performance value is reported in Table 2. Note that the time cost reported for the video example in Table 2 is the time for processing the whole video (all the frames). It could be substantially accelerated for fast previewing purposes, when users desire to see a single (or a few) frames of the video, and only the pixels on the previewing frames need to be propagated.

6. Conclusion

In this paper we have presented a novel, efficient approach to remove aliasing artifacts in edit propagation. we introduced a novel representation, the Antialias Map, to store the blending weights and relative positions of nearby interpolating pixels for each edge pixel. While previous works [1, 2, 3] directly consider edge pixels in edit propagation process, instead, we replace each edge pixel by its interpolating pixels and consider those interpolating pixels in edit propagation process. Our method is independent of a specific edit propagation algorithm and could be integrated into any existing edit propagation methods such as [1, 2, 3]. The results demonstrates that our method effectively and efficiently restores the antialiased smooth edges.

There are some works that we would like to address in the future. First, we currently deal with videos frame by frame, and for each frame we use a 2D Antialias Map. We would like to explore methods to extend Antialias Map to a 3D representation so that it could also handle motion blurs in the temporal dimension; Secondly, we would like to investigate how Antialias Map could be used for other image related applications, such as image compositing [56, 57, 58, 59] and non-photorealistic rendering [60], since it is also desired to preserve antialiased edges when compositing new images.

7. Acknowledgements

We thank all the reviewers for their valuable comments and insightful suggestions. This work was supported by the National Basic Research Project of China (Project Number 2011CB302205), the Natural Science Foundation of China (Project Number 61120106007 and 61170153).

References

[1] Xu K, Li Y, Ju T, Hu SM, Liu TQ. Efficient affinity-based edit propagation using k-d tree. ACM Transactions on Graphics 2009;28(5):118:1–118:6.

[2] Li Y, Ju T, Hu SM. Instant propagation of sparse edits on images and videos. Computer Graphics Forum 2010;29(7):2049–2054.

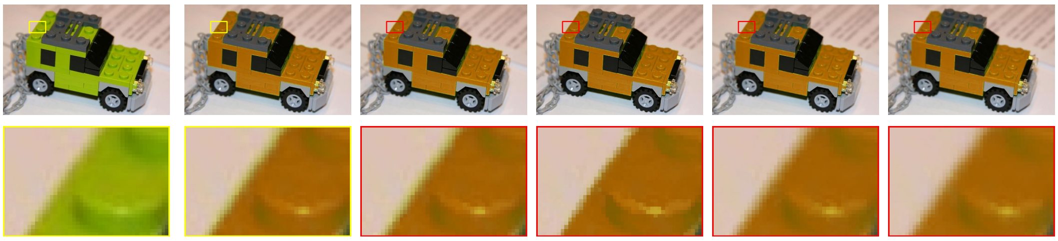
[3] An X, Pellacini F. Appprop: all-pairs appearance-space edit propagation. ACM Transactions on Graphics 2008;27(3):40:1–40:9.

[4] Yang L, Sander PV, Lawrence J, Hoppe H. Antialiasing recovery. ACM Transactions on Graphics 2011;30(3):22:1–22:9.

[5] Pérez P, Gangnet M, Blake A. Poisson image editing. ACM Transactions on Graphics 2003;22:313–318.

[6] Cheng MM, Zhang FL, Mitra NJ, Huang X, Hu SM. Repfinder: finding approximately repeated scene elements for image editing. ACM Transactions on Graphics 2010;29:83:1–83:8.

[7] Huang H, Zhang L, Zhang HC. Repsnapping: Efficient image cutout for repeated scene elements. Computer Graphics Forum 2011;30:2059–2066.



(a) source image (b) Xu et al. (c) $\sigma_a = 0.8$ (d) $\sigma_a = 0.4$ (e) $\sigma_a = 0.1$ (f) $\sigma_a = 0.0$

Figure 4: *Comparison of edit propagation results generated by Xu et al. [1] and by our method with different weight threshold σ_a. (a) is the source image O. (b) is the edit propagation result of [1], artifacts can be found along the boundaries. (c)–(f) are results using our algorithm with σ_a = 0.8, 0.4, 0.1, 0.0.*

(a) source (b) Xu et al. (c) $N = 1$ (d) $N = 2$ (e) $N = 4$ (f) $N = 8$

Figure 5: *Comparison of edit propagation results generated by Xu et al. [1] and by our method with different maximum iteration number N. (a) is the source image. (b) is the edit propagation result generated by Xu et al.. Notice that artifacts can be found along the boundaries. (c)-(f) are results using our method with N = 1, 2, 4, 8, respectively.*

[8] Zhang Y, Tong R. Environment-sensitive cloning in images. The Visual Computer 2011;27(6-8):739–48.

[9] Seol Y, Seo J, Kim PH, Lewis JP, Noh J. Weighted pose space editing for facial animation. The Visual Computer 2012;28(3):319–27.

[10] Yang F, Li B. Unsupervised learning of spatial structures shared among images. The Visual Computer 2012;28(2):175–80.

[11] Reinhard E, Ward G, Pattanaik S, Debevec P. High dynamic range imaging: Acquisition, Display, and Image-Based Lighting. 2005.

[12] Gooch AA, Olsen SC, Tumblin J, Gooch B. Color2gray: salience-preserving color removal. ACM Transactions on Graphics 2005;24:634–639.

[13] Lischinski D, Farbman Z, Uyttendaele M, Szeliski R. Interactive local adjustment of tonal values. ACM Transactions on Graphics 2006;25(3):646–653.

[14] Huang H, Xiao X. Example-based contrast enhancement by gradient mapping. The Visual Computer 2010;26:731–738.

[15] Pajak D, Čadík M, Aydın TO, Okabe M, Myszkowski K, Seidel HP. Contrast prescription for multiscale image editing. The Visual Computer 2010;26:739–748.

[16] Wu J, Shen X, Liu L. Interactive two-scale color-to-gray. The Visual Computer 2012;28(6-8):723–31.

[17] Welsh T, Ashikhmin M, Mueller K. Transferring color to greyscale images. ACM Transactions on Graphics 2002;21(3):277–280.

[18] Levin A, Lischinski D, Weiss Y. Colorization using optimization. ACM Transactions on Graphics 2004;23(3):689–694.

[19] Yatziv L, Sapiro G. Fast image and video colorization using chrominance blending. IEEE Transactions on Image Processing 2006;15(5):1120–1129.

[20] Xiao X, Ma L. Color transfer in correlated color space. In: Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications. VRCIA '06; 2006, p. 305–309.

[21] Fattal R. Single image dehazing. ACM Transactions on Graphics 2008;27:72:1–72:9.

[22] He K, Sun J, Tang X. Single image haze removal using dark channel prior. IEEE Transactions on Pattern Analysis and Machine Intelligence 2010;99:2341–2353.

[23] Ding M, Tong R. Efficient dark channel based image dehazing using quadtrees. Science China Information Science 2012;.

[24] Zhang J, Li L, Zhang Y, Yang G, Cao X, Sun J. Video dehazing with spatial and temporal coherence. The Visual Computer 2011;27(6-8):749–57.

[25] Xiao C, Gan J. Fast image dehazing using guided joint bilateral filter. The Visual Computer 2012;28(6-8):713–21.

[26] Chen J, Paris S, Durand F. Real-time edge-aware image processing with the bilateral grid. ACM Transactions on Graphics 2007;26(3).

[27] Li Y, Adelson E, Agarwala A. Scribbleboost: Adding classification to edge-aware interpolation of local image and video adjustments. Computer Graphics Forum 2008;27(4):1255–1264.

[28] Fattal R, Carroll R, Agrawala M. Edge-based image coarsening. ACM Transactions on Graphics 2009;29(1):6:1–6:11.

[29] Fattal R. Edge-avoiding wavelets and their applications. ACM Transactions on Graphics 2009;28(3):22:1–22:10.

[30] Criminisi A, Sharp T, Rother C, P'erez P. Geodesic image and video editing. ACM Transactions on Graphics 2010;29:134:1–134:15.

[31] Xie ZF, Lau RWH, Gui Y, Chen MG, Ma LZ. A gradient-domain-based edge-preserving sharpen filter. The Visual Computer 2012;.

Data		toy (Fig. 4)	flower (Fig. 5)	cake (Fig. 7)	dog (Fig. 7)	branch (Fig. 6)	parrot (Fig. 6)	sky (Fig. 1)	bird (Fig. 8)
Type		image	image	image	image	image	image	image	video
Resolution		120K	120K	120K	120K	150K	150K	240K	30M
Frame Num		-	-	-	-	-	-	-	400
K-d tree	time	22ms	23ms	17ms	25ms	28ms	24ms	41ms	8s
	memory	8MB	8MB	8MB	8MB	8MB	8MB	8MB	22MB
Improved k-d tree	time	40ms	42ms	32ms	45ms	45ms	47ms	79ms	13s
	memory	9MB	9MB	9MB	9MB	9MB	9MB	9MB	24MB
RBF	time	16ms	17ms	13ms	20ms	21ms	19ms	26ms	4s
	memory	1MB	1MB	1MB	1MB	1MB	1MB	1MB	1MB
Improved RBF	time	32ms	30ms	25ms	38ms	32ms	36ms	51ms	8s
	memory	1MB	1MB	1MB	1MB	1MB	1MB	1MB	1MB

Table 2: *Performance comparison between the k-d tree method [1], our method combined with the k-d tree approach, RBF method [2] and our method combined with the RBF method. Both running time and memory cost are reported.*

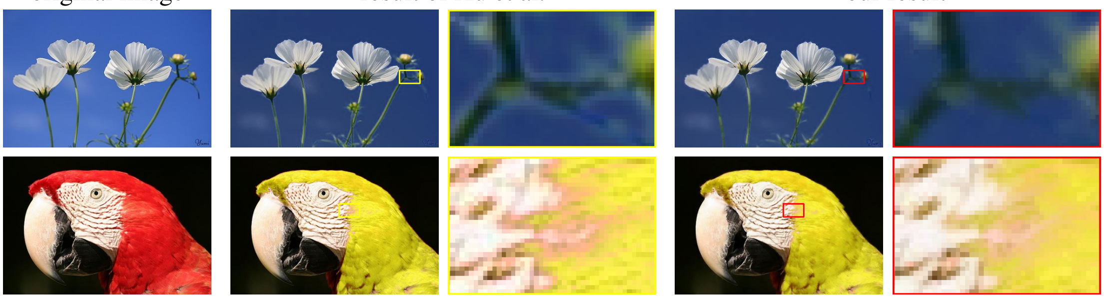


Figure 6: *Results generated by Xu et al. [1] and by our method. The first column give the original images; the second and third columns are results generated by Xu et al.; the fourth and fifth columns are results generated by our method.*

[32] Pellacini F, Lawrence J. Appwand: editing measured materials using appearance-driven optimization. ACM Transactions on Graphics 2007;26(3):54–54.

[33] Xiao C, Nie Y, Tang F. Efficient edit propagation using hierarchical data structure. IEEE Transactions on Visualization and Computer Graphics 2011;17:1135–1147.

[34] Bie X, Huang H, Wang W. Real time edit propagation by efficient sampling. Computer Graphics Forum 2011;30(7):2041–2048.

[35] Dong Y, Wang J, Tong X, Snyder J, Lan Y, Ben-Ezra M, et al. Manifold bootstrapping for svbrdf capture. ACM Transactions on Graphics 2010;29:98:1–98:10.

[36] Lan Y, Dong Y, Wang J, Tong X, Guo B. Condenser-based instant reflectometry. Computer Graphics Forum 2010;29:2091–2098.

[37] Ren P, Wang J, Snyder J, Tong X, Guo B. Pocket reflectometry. ACM Transactions on Graphics 2011;30:45:1–45:10.

[38] Dong Y, Tong X, Pellacini F, Guo B. Appgen: Interactive material modeling from a single image. ACM Transactions on Graphics 2011;30(6):146:1–146:10.

[39] Dana KJ, van Ginneken B, Nayar SK, Koenderink JJ. Reflectance and texture of real-world surfaces. ACM Transactions on Graphics 1999;18(1):1–34.

[40] Xu K, Wang J, Tong X, Hu SM, Guo B. Edit propagation on bidirectional texture functions. Computer Graphics Forum 2009;28(7):1871–1877.

[41] Farbman Z, Fattal R, Lischinski D. Diffusion maps for edge-aware image editing. ACM Transactions on Graphics 2010;29:145:1–145:10.

[42] Pharr M, Humphreys G. Physically Based Rendering: From Theory to Implementation. Morgan Kaufmann; 2004.

[43] Akenine-möller T, Haines E, Hoffman N. Real-Time Rendering 3rd ed. AK peters; 2008.

[44] Reeves WT, Salesin DH, Cook RL. Rendering antialiased shadows with depth maps. SIGGRAPH Computer Graphics 1987;21(4):283–291.

[45] Brabec S, peter Seidel H. Hardware-accelerated rendering of antialiased shadows with shadow maps. In: Computer Graphics International. 2001, p. 209–214.

[46] Cant RJ, Shrubsole PA. Texture potential mip mapping, a new high-quality texture antialiasing algorithm. ACM Transactions on Graphics 2000;19(3):164–184.

[47] Jon PE, Marcus DW, Martin W, Paul FL. Implementing an anisotropic texture filter. Computers and Graphics 2000;24(2).

[48] Pan M, Wang R, Chen W, Zhou K, Bao H. Fast, sub-pixel antialiased shadow maps. Computer Graphics Forum 2009;28:1927–1934.

[49] Van-hateren JH, Vander-schaaf A. Independent component filters of natural images compared with simple cells in primary visual cortex. Proceedings of the Royal Society 1998;B(265):359–366.

[50] Hyvärinen A, Hurri J, Hoyer PO. Natural image statistics: A probabilistic approach to early computational vision. Springer; 2009.

[51] Reshetov A. Morphological antialiasing. In: Proceedings of the Conference on High Performance Graphics 2009. HPG '09; 2009, p. 109–116.

[52] Zhang SH, Chen T, Zhang YF, Hu SM, Martin RR. Vectorizing cartoon animations. Visualization and Computer Graphics 2009;15(4):618–629.

[53] Lai YK, Hu SM, Martin RR. Automatic and topology-preserving gradient mesh generation for image vectorization. ACM Transactions on Graphics 2009;28(3):85:1–85:8.

[54] Johannes K, Dani L. Depixelizing pixel art. ACM Transactions on Graphics 2011;30:99:1–99:8.

[55] Ma LQ, Xu K. Antialiasing recovery for edit propagation. In: Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry. VRCAI '11; 2011, p. 125–130.

[56] Chen T, Cheng MM, Tan P, Shamir A, Hu SM. Sketch2photo: internet image montage. ACM Transactions on Graphics 2009;28:124:1–124:10.

[57] Ding M, Tong RF. Content-aware copying and pasting in images. The Visual Computer 2010;26:721–729.

[58] Huang H, Zhang L, Zhang HC. Arcimboldo-like collage using internet images. ACM Transactions on Graphics 2011;30(6):155:1–155:8.

[59] Du H, Jin X. Object cloning using constrained mean value interpolation. The Visual Computer 2012;.

[60] Huang H, Fu T, Li CF. Painterly rendering with content-dependent natural paint strokes. The Visual Computer 2011;27(9):861–71.

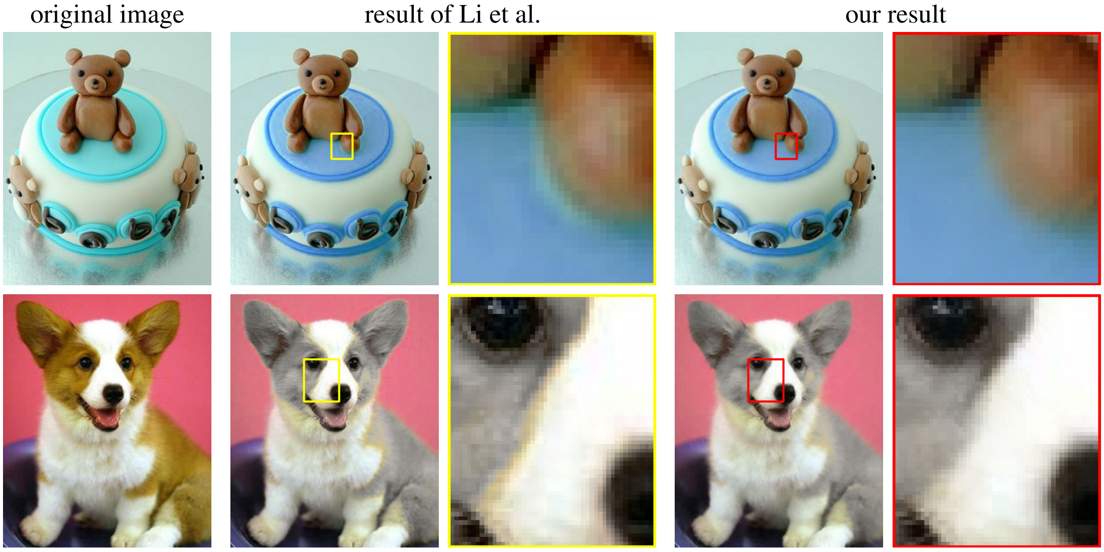


Figure 7: *Results generated by Li et al. [2] and by our method. The first column give the original images; the second and third columns are results generated by Li et al.; the fourth and fifth columns are results generated by our method.*

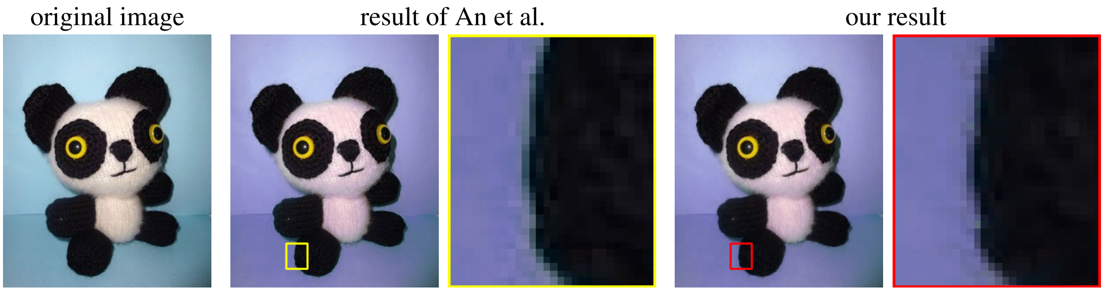


Figure 8: *Results generated by An et al. [3] and by our method. The first column give the original images; the second and third columns are results generated by An et al.; the fourth and fifth columns are results generated by our method.*

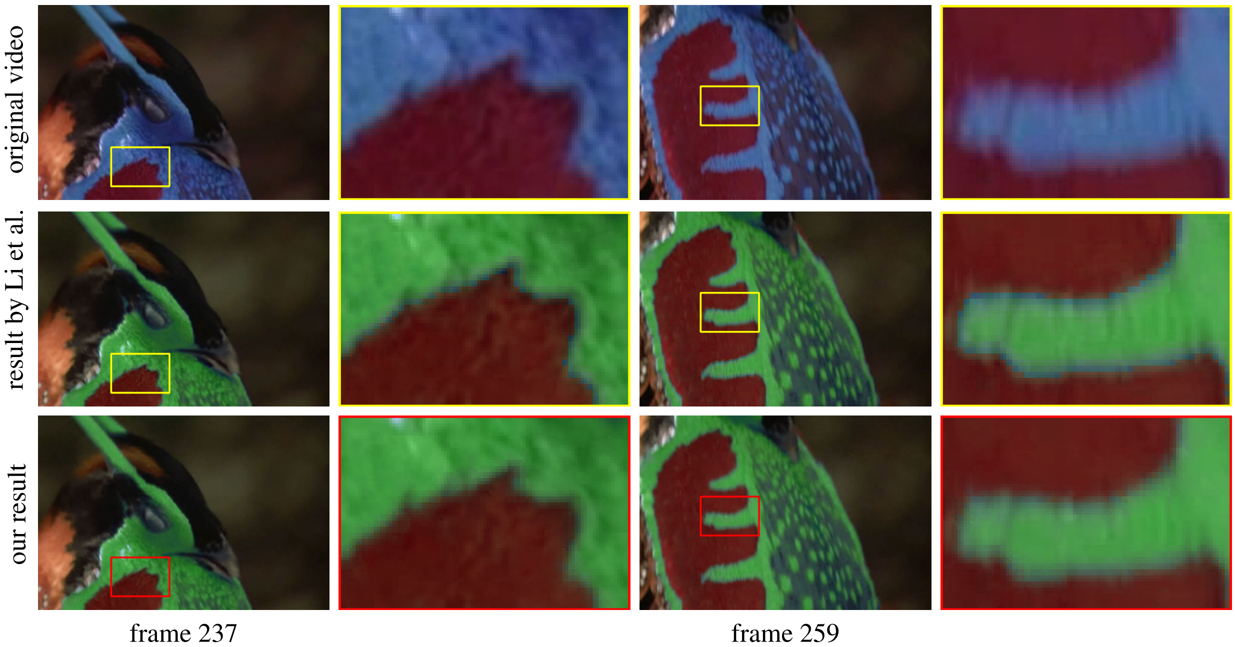


Figure 9: *Video results generated by Li et al. and our method. We have shown two frames of the video and clearly our method improves a lot along the boundaries.*