AVOIDING BLEEDING IN IMAGE BLENDING

Minxuan Wang⋆ *Zhe Zhu*⋆ *Songhai Zhang*⋆ *Ralph Martin*† *Shi-Min Hu*⋆

⋆ TNList, Tsinghua University
† School of Computer Science and Informatics, Cardiff University

ABSTRACT

Though elegant in mathematical formulation, gradient-domain image blending suffers from bleeding artefacts in real world applications. We propose an image blending algorithm that avoids bleeding artefacts while preserving the good properties of gradient-domain blending, such as smooth transitions between the candidate regions. Our key idea to finesse the non-smooth boundary difference calculation that causes bleeding artefacts is to use local patch differences. While most previous gradient-domain blending algorithms change one region to fit the other, to further reduce bleeding, we perform bidirectional blending so that both regions change simultaneously. Our blending algorithm is fast: when applied to image stitching, it can achieve 20 fps at 4K resolution. Source code and test images are publicly available[1].

***Index Terms*—** gradient-domain blending, bleeding artefacts, image stitching.

1. INTRODUCTION

Blending is a widely used technique in image editing[1, 2], used for example in copying and pasting of objects, and in image stitching. Image blending typically changes one region, the *source region*, to fit the other, the *target region*. Gradient-domain image blending is a popular approach. It diffuses differences along the boundary between the regions to the entire region to be blended. In 2003, Perez et al. [3] showed how to formulate blending in terms of solving a Poisson equation. Poisson blending is mathematical elegant, but suffers from two severe drawbacks. Firstly, it is time consuming, as it needs to solve a large linear equation; the number of unknowns is the number of blended pixels. Secondly, it suffers from bleeding artefacts when the blending boundary is not smooth. To solve the first problem, various gradient-domain blending algorithms have been proposed based on the original Poisson blending approach. Agarwala et al. [4] observed that as the difference between the unblended source region and blended source region is smooth, the offset map can be approximated by a quadtree representation. Now, the number of unknowns is equal to the number of quadtree vertices, leading to a significantly smaller linear system. Another way to reduce the number of unknowns is to use splines [5] to approximate the offset map. Tanaka et al. [6] modified the original Poisson blending approach by adding a color preservation term, and then solving the equation in the frequency domain. The final blended result has smooth transitions and preserves the original colors as much as possible. Farbman et al. [7] showed that it is possible to avoid solving the linear system, instead using mean value coordinates to interpolate the offset map. As noted in[8], this method also suffers from bleeding artefacts. This process can be made faster still by sophisticated use of convolution operations [9]. We later refer to the former as *mean-value coordinates* (MVC) blending and the latter as *convolution pyramid* (ConvPyr) blending. Both methods are fast and parallelizable, but still suffer from bleeding artefacts.

To solve the bleeding problem in gradient-domain blending, Jia et al. [10] suggested optimizing the boundaries of the source region. They designed an energy function to encourage the boundaries to go through positions where the difference between source and target region pixels is nearly constant. This works well for the object insertion problem; Summa et al. [11] used a similar strategy for panorama stitching. Instead of modifying the blending boundary, another way to reduce bleeding artefacts is to control the interpolation process [12] so that bleeding occurs most in textured regions where the human eye is less able to perceive it. However, all of the above mentioned methods are data dependent, and are unsuitable for real world applications as they are not robust, and are slow.

In this paper, we propose a gradient-domain blending algorithm which avoids bleeding, for use in image stitching. Traditional gradient-domain blending algorithms usually calculate boundary differences along a band that is only 1 pixel wide. Due to misalignment of the images to be stitched, or random noise, the calculated boundary differences are not always accurate, which can cause bleeding artefacts. Our algorithm is based on two observations. The first is that the difference at a boundary point calculated relative to a local region is more stable than one calculated from a single position. Using this approach, the obtained boundary differences vary more smoothly. The second observation is that large boundary differences lead to much more obvious bleeding artefacts. Instead of changing one region to fit the other, we let both regions change. In this way bleeding can be diffused into

[1] https://github.com/hitminxuanwang/DeBleeding

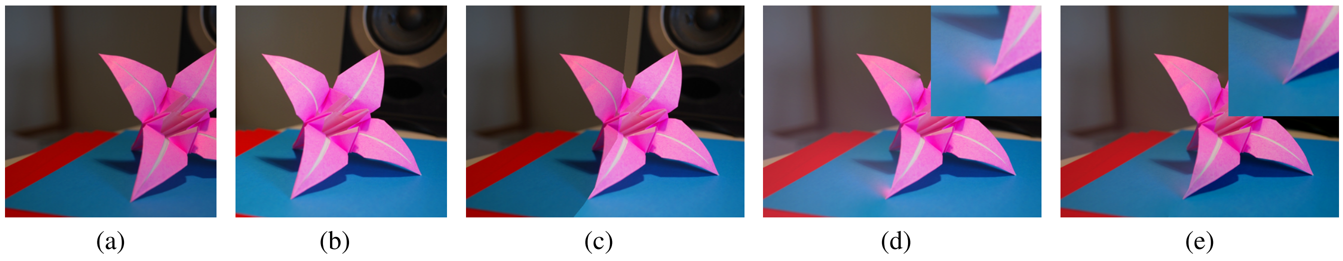

Fig. 1. (a,b): two overlapping input images. (c): stitched result without blending. (d): blending result using ConvPyr. (e): blending result using our method.

separate regions.

In the following, we briefly summarise MVC and ConvPyr blending algorithms, and then give the details of our proposed algorithm in Section 2. Experimental results are demonstrated in Section 3 and draw conclusions in Section 4.

2. ALGORITHM

Our method aims to avoid bleeding artefacts in gradient-domain blending for image stitching. We first briefly summarise the most efficient gradient-domain blending algorithms–MVC and ConvPyr blending, and then explain our approach to avoiding bleeding artefacts in them. Since ConvPyr is based on MVC blending, we first introduce the latter.

2.1. Problem Formulation

We assume that the inputs are two overlapping images, as in Figure 1(a,b). These two images are first stitched using e.g. the algorithm in [13]. This initial result is shown in Figure 1(c). If ConvPyr blending is used, it gives the result in Figure 1(d). Our blending method gives the improved result in Figure 1(e).

In stitching, the overlap area has approximately the same content in both images, but there may be lighting differences. The goal of blending is to compute an offset map giving values to be added to the unblended pixel values. Given that the offset map is a smooth field with fixed boundary values, this smooth field can be approximated using mean value coordinate interpolation:

$$r(x) = \sum_{i=0}^{m-1} \lambda_i(x) b(p_i) \tag{1}$$

where $r(x)$ is the offset field at position x, m is the number of boundary points, p_i is the position of the i^{th} boundary point, $b(p_i)$ is the boundary difference at position p_i, and $\lambda_i(x)$ is the mean value coordinate at x for the i^{th} boundary point. See Figure 2.

The mean value coordinates in Equation 1 are given by:

$$\lambda_i(x) = \omega_i / \sum_{j=0}^{m-1} \omega_j, \qquad i = 0, \dots, m-1, \tag{2}$$

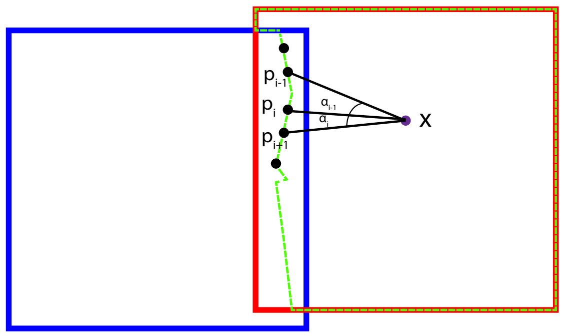

Fig. 2. MVC interpolation. Blue rectangle: target image. Red rectangle: source image. Green dotted line: target region boundary, calculated by graph cut. The offset value at x is calculated using differences at boundary points, and corresponding mean value coordinates. The MVC value at x for boundary point p_i depends on the angles between lines xp_{i-1}, xp_i and the angle between lines xp_i and xp_{i+1}.

$$\omega_i = \left(\tan(\alpha_{i-1}/2) + \tan(\alpha_i/2)\right) / \|p_i - x\|, \tag{3}$$

where α_i is the angle between lines xp_i and xp_{i+1}, and α_{i+1} is defined in a similar way.

ConvPyr blending is based on MVC blending. Substituting Equation 2 into Equation 1 gives the following expression

$$r(x) = \sum_i \omega_i(x) b(p_i) / \sum_i \omega_i(x) \tag{4}$$

Writing it in this way, the interpolation operation can be thought as a division of convolution operations. Since the convolution kernel is large, this is time consuming. In ConvPyr blending, the convolution operation with a large kernel is approximated by convolution with several small kernels, significantly reducing the computation time. Our algorithm is based on ConvPyr blending, and we focus on use of a robust boundary calculation to avoid bleeding artefacts.

2.2. Proposed Algorithm

It is well known that bleeding artefacts in gradient-domain blending are caused by lack of smoothness in boundary piexel

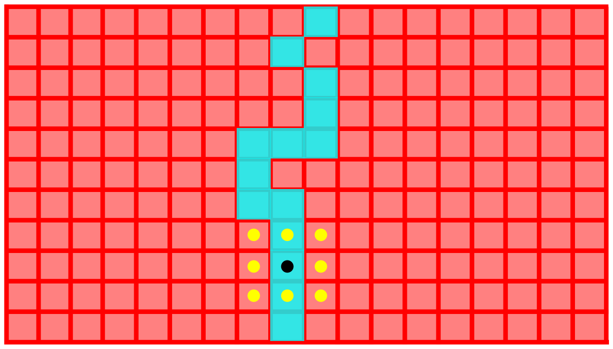

Fig. 3. Local difference calculation. Blue pixels: boundary pixels. Red pixels: left, target region, right source region. Black dot: pixel whose boundary difference is being calculated. Yellow dots: 3×3 local patch over which differences are calculated and averaged (in practice we use 7×7 patches).

value differences. Our approach is based on two key observations. The first is that misalignment of the input images, or noise, may cause the boundary differences calculated along a one-pixel wide boundary to lack smoothness. Calculating these boundary differences using a local region for each pixel can increase their smoothness. The second observation is that less boundary differences can cause much unobvious bleeding. Thus, instead of changing one region to fit the other, we change both regions to meet a compromise. These two improvements can dramatically reduce bleeding artefacts in image stitching, so that in many cases the reduced bleeding is visually imperceptible.

2.2.1. *Local Differences*

Our key idea is to improve the boundary difference calculation. In ConvPyr blending (and most other gradient-domain blending methods) the boundary difference $b(p_i)$ is calculated as:

$$b(p_i) = I_t(p_i) - I_s(p_i) \tag{5}$$

where $I_t(p_i)$ and $I_s(p_i)$ are pixel values in target and source images at position p_i respectively. Instead, we use the mean difference of a local patch:

$$b(p_i) = \text{mean}(N_t(p_i) - N_s(p_i)) \tag{6}$$

where $N_t(p_i)$ and $N_s(p_i)$ are square patches centered at position p_i in the target and source images respectively. $mean()$ denotes the mean operation. See Figure 3. In our implementation we usually use 7×7 patches. While such boundary differences may not be as accurate as pixel-wise values, they more robust, and smoother.

Case	Pixels	MVC	ConvPry	Ours
Fig. 1	0.25 Mpix	101ms	107ms	110ms
Fig. 4	4 Mpix	827ms	1247ms	1304ms
Fig. 5	8 Mpix	2535ms	4782ms	4903ms

Table 1. Speed of several blending algorithms of CPU version. Pixels: number of pixels in the source region. Right 3 columns: time taken by various blending algorithms.

Case	Pixels	MVC	ConvPry	Ours
Fig. 1	0.25 Mpix	5ms	5ms	5ms
Fig. 4	4 Mpix	14ms	27ms	28ms
Fig. 5	8 Mpix	31ms	63ms	67ms

Table 2. Speed of several blending algorithms of GPU version. Pixels: number of pixels in the source region. Right 3 columns: time taken by various blending algorithms.

2.2.2. *Two-sided Diffusion*

To further reduce the degree of bleeding, we let both the source region and target region change as it allows the boundary differences to be diffused more effectively. Let the boundary used for changing the source region to fit the target region be the *source-target boundary*. We define a *target-source boundary* by moving the *source-target boundary* one pixel towards the target. Equation 4 can now be rewritten as

$$r(x) = \begin{cases} \beta \frac{\sum_i \omega_i(x) b(p_i^t)}{\sum_i \omega_i(x)}, & x \in \text{Target} \\ (1-\beta) \frac{\sum_j \omega_j(x) b(p_j^s)}{\sum_j \omega_j(x)}, & x \in \text{Source} \end{cases} \tag{7}$$

where p_i^t and p_i^s are boundary points in the *target-source boundary* and *source-target boundary* respectively. We set β to 0.5 in our implementation.

3. EXPERIMENTS

Our approach has been implemented in C++ on a PC with an Intel i7 3770 CPU, a NVIDIA GTX 970 GPU and 16GB RAM. We consider the speed as well as blending results of our proposed method.

3.1. Speed

We have compared the speed of our proposed algorithm with MVC blending and ConvPyr blending. Results of CPU version and GPU version are given in Table 1 and Table 2, which show that our proposed blending algorithm is efficient even for large number of pixels. The GPU version is implemented using CUDA.

3.2. Quality

We have evaluated the quality of the results of our approach on a various real-world examples, including both images and

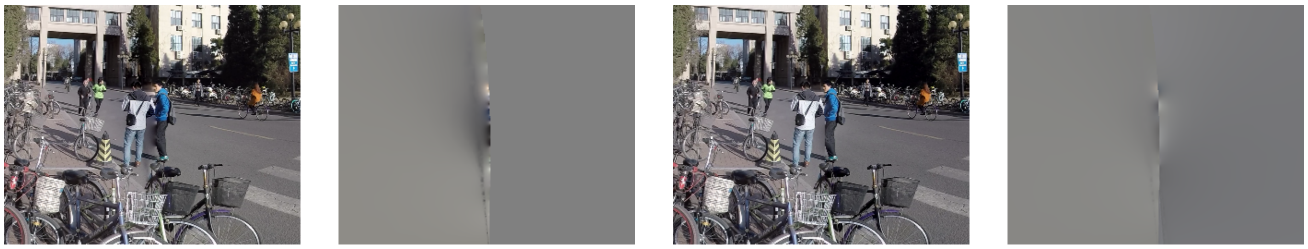

Fig. 4. Blending results for an outdoor scene. (a): ConvPyr result, (b): ConvPyr offset map, (c): our result, (d): our offset map.

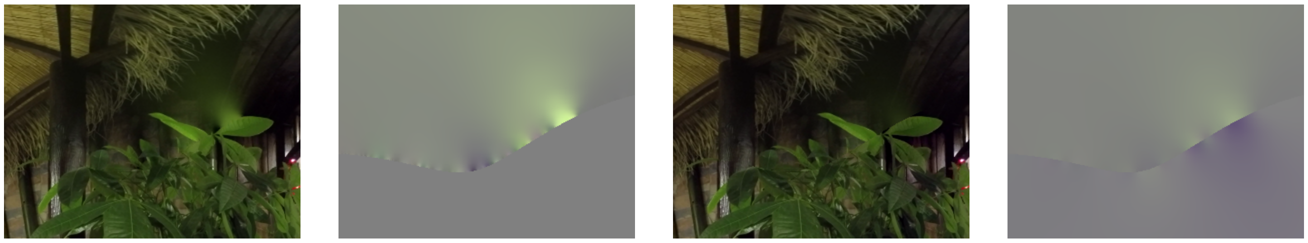

Fig. 5. Blending results for an indoor scene. (a): ConvPyr result, (b): ConvPyr offset map, (c): our result, (d): our offset map.

Case	MVC	ConvPry	Ours
Fig. 1	37.20	39.33	16.07
Fig. 4	11.54	9.60	0.47
Fig. 5	24.77	26.67	5.56

Table 3. Bleeding degree of MVC blending, ConvPyr blending and our approach. Compared with those two methods, our approach has a significant lower bleeding degree.

videos. An outdoor scene is shown in Figure 4 while an indoor scene is shown in Figure 5(They are cropped from their original size). Then we follow the image blending quality assessment method proposed in [14] to evaluate the results of our approach. To quantify the degree of bleeding, we first calculate an energy map by calculating the absolute values of the offset map:

$$EM(i) = |p_i - \bar{p}|, \tag{8}$$

where $\bar{p}$ at each pixel is the averaged value of the offset map over the whole image. p_i is the pixel value at location i. We next calculate the bleeding map based on the energy map:

$$B(i) = \max(0, p_i - \alpha \frac{E_h}{A_h + \delta}). \tag{9}$$

In the above equation A_h is the number of non-zero values of the binarized energy map using Otsu's method [15], and E_h is the sum of the energy of the non-zero positions on the binarized map. α is a weight and we set it to 2 in our experiment. This weight is used to truncate with a high peak value. Given the bleeding map, we define the bleeding degree D as the sum over the squared values in each position of the bleeding map:

$$D = \sum B(i)^2, \tag{10}$$

We compared our result with MVC blending and ConvPyr blending using bleeding degree as a metric. Results are given in Table 3. They demonstrate that our approach better avoids bleeding artefacts. Video examples are presented in the supplemental material.

4. CONCLUSION AND DISCUSSION

This paper has proposed two improvements to gradient-domain blending to avoid bleeding artefacts. Our approach is fast and works well in most real world cases, although at times small artefacts remain. A potential way to resolve such issues is to detect such *bad regions* that cause the bleeding, and to perform blending after correcting those regions.

5. ACKNOWLEDGEMENT

This work was supported by the Natural Science Foundation of China (Project Number 61521002, 61373069), Research Grant of Beijing Higher Institution Engineering Research Center, and Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology.

6. REFERENCES

[1] Tao Chen, Zhe Zhu, Ariel Shamir, Shi-Min Hu, and Daniel Cohen-Or, "3-sweep: Extracting editable objects

from a single photo," *ACM Trans. Graph.*, vol. 32, no. 6, pp. 195:1–195:10, Nov. 2013.

[2] Tao Chen, Zhe Zhu, Shi-Min Hu, Daniel Cohen-Or, and Ariel Shamir, "Extracting 3d objects from photographs using 3-sweep," *Commun. ACM*, vol. 59, no. 12, pp. 121–129, Dec. 2016.

[3] Patrick Pérez, Michel Gangnet, and Andrew Blake, "Poisson image editing," in *ACM Transactions on Graphics (TOG)*. ACM, 2003, vol. 22, pp. 313–318.

[4] Aseem Agarwala, "Efficient gradient-domain compositing using quadtrees," in *ACM Transactions on Graphics (TOG)*. ACM, 2007, vol. 26, p. 94.

[5] Richard Szeliski, Matthew Uyttendaele, and Drew S-teedly, "Fast poisson blending using multi-splines," in *Computational Photography (ICCP), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–8.

[6] Masayuki Tanaka, Ryo Kamio, and Masatoshi Okutomi, "Seamless image cloning by a closed form solution of a modified poisson problem," in *SIGGRAPH Asia 2012 Posters*. ACM, 2012, p. 15.

[7] Zeev Farbman, Gil Hoffer, Yaron Lipman, Daniel Cohen-Or, and Dani Lischinski, "Coordinates for instant image cloning," in *ACM Transactions on Graphics (TOG)*. ACM, 2009, vol. 28, p. 67.

[8] Z. Zhu, H. Z. Huang, Z. P. Tan, K. Xu, and S. M. Hu, "Faithful completion of images of scenic landmarks using internet images," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 8, pp. 1945–1958, Aug 2016.

[9] Zeev Farbman, Raanan Fattal, and Dani Lischinski, "Convolution pyramids.," *ACM Trans. Graph.*, vol. 30, no. 6, pp. 175, 2011.

[10] Jiaya Jia, Jian Sun, Chi-Keung Tang, and Heung-Yeung Shum, "Drag-and-drop pasting," *ACM Transactions on Graphics*, 2006.

[11] Brian Summa, Julien Tierny, and Valerio Pascucci, "Panorama weaving: fast and flexible seam processing," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 83:1–83:11, July 2012.

[12] Michael W. Tao, Micah K. Johnson, and Sylvain Paris, "Error-tolerant image compositing," *International Journal of Computer Vision*, vol. 103, no. 2, pp. 178–189, 2013.

[13] Richard Szeliski, "Image alignment and stitching: A tutorial," *Found. Trends. Comput. Graph. Vis.*, vol. 2, no. 1, pp. 1–104, Jan. 2006.

[14] Zhe Zhu, Jiaming Lu, Minxuan Wang, Song-Hai Zhang, Ralph R. Martin, Hantao Liu, and Shi-Min Hu, "A comparative study of blending algorithms for realtime panoramic video stitching," *CoRR*, vol. abs/1606.00103, 2016.

[15] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, Jan 1979.