A Comparative Study of Blending Algorithms for Realtime Panoramic Video Stitching

Zhe Zhu\textsuperscript{1} * Jiaming Lu\textsuperscript{1} Songhai Zhang\textsuperscript{1} Ralph Martin\textsuperscript{2} Hantao Liu\textsuperscript{2} Shimin Hu\textsuperscript{1}

\textit{Tsinghua University\textsuperscript{1} Cardiff University\textsuperscript{2}}

June 2, 2016

Abstract

Panoramic video stitching consists of two major steps: remapping each candidate video stream to its final position and compositing them to generate seamless results. Given videos captured with cameras in fixed relative positions, the remapping step can be done directly using a precomputed look-up table. Greater challenges lie in the more time-consuming composition step. Real world applications typically use blending to perform composition; the performance of the whole system largely depends on the efficiency of the blending algorithm. In this paper, we provide in-depth analysis of the application of several state-of-the-art image blending techniques to \textit{realtime} panoramic video stitching, as \textit{realtime} panoramic video stitching enables near-immediate broadcast. Test videos were captured under various conditions, and stitched using various blending methods. Both computational efficiency and quality of composition results were evaluated. Source code and test videos are all publicly available.

1 Introduction

We are witnessing a boom in popularity of virtual reality (VR) techniques due to a new generation of display devices such as Oculus Rift, Sony Morpheus and HTC Vive, which are bringing immersive virtual worlds to many users. There are two main sources of VR content: rendered 3D scenes and captured panoramic stereoscopic videos. The former are often used in VR games, while the latter have a wider range of application scenarios. For example, in a live VR football match, a noticeable delay would unbearable for football fans when nearby friends might get results in realtime via other services. Even if a delay is acceptable in other applications such as VR news, there is still an overall requirement to stitch frames at the same rate as they are produced for near-immediate broadcast.

*ajex1988@gmail.com
As an important source of VR content, panoramic videos can be obtained by stitching multiple overlapping ordinary videos. Stitching panoramic videos requires stitching synchronized videos frame by frame. Though panoramic image stitching has been studied in depth [1], most attention has been paid to the visual quality of the results rather than to the computational efficiency. Since panoramic videos typically have tens of high resolution frames per second, efficiency of blending algorithms is an important issue for video panorama stitching.

Panoramic video stitching consists of two main steps: remapping each candidate video stream to its final position, and compositing them to generate seamless results. Given videos captured with cameras in fixed relative positions, the remapping step can be done directly using a precomputed look-up table. Greater challenges lie in the more time-consuming composition step. Real world applications typically use blending to perform composition. In this paper, we provide in-depth analysis of several state-of-the-art image blending techniques with application to realtime panoramic video stitching. As some blending algorithms can readily be parallelized, we also consider GPU versions of these algorithms.

We assume that boundary seams for overlapping videos are fixed. One reason is that fixing boundary seams helps to avoid the flicker among different frames. While different successive frames may have different optimal seams, directly using different optimal seams for each frame can lead to degradation of coherence. The other reason is that fixed seams allows more to be pre-computed, such as mean value coordinates (MVC) if MVC blending is to be used.

In the past few decades, many blending algorithms have been proposed [2, 3, 4, 5, 6]. While these methods have been carefully compared with respect to panoramic image blending, their comparative benefits in panoramic video stitching have not been adequately investigated.

In this paper, we describe a panoramic video stitching system, and use it to compare our own implementations of several state-of-the-art blending algorithms, using GPU implementations where possible. Stitching without blending is also used as a baseline for comparison. We have captured various kinds of scenes and use them to evaluate the effectiveness and efficiency of the blending algorithms. Our code and test videos are all publicly available.¹

2 Related Work

Image blending has long been studied by the image processing and computer graphics research communities. Perhaps the most widely used approach, due in part to availability of open source is multi-band blending [5]. It is easy to implement and provides stable blending results. It constructs a Laplacian pyramid and blends the images at each level. The final result is obtained by merging all images at each level. Perez et al [2] formulate image blending via a Poisson equation whose solution can be obtained by solving a large sparse linear system.

¹URL TO BE ADDED AFTER REFEREEING.
Although this formulation is mathematically elegant and the blending results are perfect when the boundary is smooth, it is time consuming, especially for large images. It also suffers from bleeding artefacts if the blending boundary is not sufficiently smooth. Agarwala [3] observes that the offset between the original content and the blended content in the target region is piecewise smooth, allowing ready approximation of the whole offset field by a quadtree. This representation significantly reduces the number of variables in the linear system, accelerating the blending procedure. Szeliski et al [6] further observe that if we provide each image with a separate offset field represented by a low-dimensional spline, each offset field becomes everywhere smooth rather than piecewise smooth. As the spline has low dimensionality, the number of variables can be further reduced. To avoid solving linear equations, Farbman et al [4] instead use mean-value coordinates (MVC) to interpolate the smooth offset field from boundary differences. Given a target region of fixed shape, the coordinates can be pre-computed and re-used for all frames. Another advantage of this method is that it is fully parallelizable. Since it provides an approximation of the Poisson formulation, it suffers from bleeding artefacts like other Poisson-based methods.

To overcome such artefacts, Jia et al. [7] first optimize a smooth boundary derived from the initial boundary, and perform Poisson blending on the optimized boundary. An alternative way to avoid artefacts is to assign a weight map to the image so that the errors that causes bleeding are shifted to textured regions, where the consequences are less obvious to the human eye [8]. Unfortunately, both of these methods are time consuming, and cannot ensure interframe coherence, so neither of those methods are suitable for realtime panoramic video stitching.

3 System Overview

To capture various indoor scenes and outdoor scenes for algorithm testing and analysis, we built a camera rig which could be supported in various ways. We first described the platform we used to capture the videos used in our tests. It is built upon a six GoPro camera rig as illustrated in Figure 1. Five cameras are arranged symmetrically in a plane around a vertical axis, while one camera is directed vertically. Using an appropriate resolution setting, neighbouring cameras provide video stream which overlap suitably. During capture, we use a GoPro Smart Remote to synchronize all cameras. Having captured a sequence, we use the PTGui tool to compute a remapping template which defines the positional relationships between the pixels in each stream, and those in the final panorama. We illustrate a stitched panorama with boundary seams and indexed source streams in Figure 2.

While a typical panorama stitching pipeline [9] consists of many processing modules, we simplify them into two main modules. For example, the results of certain other modules such as lens distortion correction can be incorporated in the remapping. Using a fixed rig allows us to precompute the remapping, and make use of it in all blending algorithms. For convenience we ignore other
Figure 1: Capture device. Left: camera rig. Center: rig mounted on a car. Right: rig mounted on a tripod.

Figure 2: A typical stitched panorama. Region 0 is captured by the upwards-pointing camera. Regions 1–5 are captured by the other cameras. Red lines indicate the fixed boundary seams between neighbouring video streams.

sophisticated modules such as exposure compensation; blending can do most of its work provided that the cameras are more or less identical.

4 Benchmark

To evaluate the performance of various blending algorithms, we have created a benchmark containing test videos captured under differing conditions. The resolution of each original captured videos is $1920 \times 1440$ and the frame rate is 30 fps.

We have separated the videos into different subsets according to three properties: illuminance conditions, camera rig motion, and object distance, as these three properties greatly affect blending results. The illuminance conditions cover four typical cases: one indoor scene with adequate lighting, one poorly-lit indoor scene, one outdoor scene with adequate lighting and one poorly-lit outdoor scene. Since the cameras are typically used in automatic exposure mode, changes in illuminance conditions have a strong effect on the brightness of the videos.
We also provide two typical types of camera motion: camera rig motion leads to content change along the boundary seams. The simple case holds the camera rig fixed. The difficult case includes large motions of the camera rig (e.g. the camera rig is mounted on a moving vehicle). Object distances also affect blending results as varying them typically affects the bleeding artefacts which arise near the boundary seam. Since human eyes are more sensitive to large objects, objects near the camera always have larger saliency and their artefacts are often more obvious. We consider three typical scenes with different object distances. The easiest case concerns moving objects far from the camera, while the most difficult case concerns moving objects near the camera.

### 5 Blending Algorithms

We analyze four blending algorithms which have been chosen as being representative for the following reasons. Feather blending has the lowest computational expense (apart from trivially clipping the images against each other), and generally provides a basic degree of visual quality. Multi-band blending is the most widely used approach in the open source community, and is insensitive to misalignment. MVC blending can be readily parallelized and also avoids solving a large linear equation, yet results in nearly no visual differences from standard Poisson blending. Multi-spline blending is the fastest Poisson-based blending algorithm. We now describe these algorithms in more detail.

**Feather Blending:** This is simply a linear combination of the source region and target region. The most straightforward approach is to set the weight to 0.5 across the entire blended region; a more reasonable approach is to use the OpenCV Feather blending class which uses a spatially varying weight map. The two blending sources are given equal weights at the boundary, which then fall off the further we go into the opposite region, until they become zero.

**Multi-band Blending:** This can be regarded as performing feather blending on images of different frequencies. It first constructs a Laplacian pyramid and linearly combines the source and target regions at different levels. The final blending result is obtained by adding all the blended images from the different levels. OpenCV provides an efficient CPU implementation of multi-band blending, to which we just make a minor change. Since the image mask of each image to be blended is fixed, the Gaussian weight pyramids can be pre-computed. We also have implemented a GPU version. Since 6 high resolution input videos are to be processed, the time needed to transfer data from PC memory to GPU memory can be significant. We are careful to do this data transfer once only. The Laplacian pyramid can be constructed in parallel using equivalent weighting functions [5]. As each level of the pyramid can be regarded as a function of the original image, we precompute this function mapping between the original image and other levels. Using all these functions we can compute each level of the pyramid simultaneously. Combination of the Laplacian images using a Gaussian weight image is also fully parallelizable.

**MVC Blending:** This approximates the Laplacian membrane used in Pois-
son blending. It constructs a harmonic interpolant given the boundary intensity differences. Unlike Poisson blending which solves for the final pixel values directly, MVC blending computes an offset map and the final blended result is obtained by adding this offset map to the region to be blended. For each pixel in the target region the offset value is a weighted linear combination of the boundary differences using a combination weight derived from the pixel’s mean value coordinate. As the boundary seams have fixed locations, the mean value coordinates and weights can be pre-computed once for all frames. Note that multi-band blending is a symmetric algorithm, as all the candidate images are linearly combined at each level, but MVC blending changes colors to match each other so a blending order needs to be chosen in advance: source and target are not treated symmetrically. Since the final value of each pixel in the target region only depends on its mean value coordinate and the boundary offset, final pixel values are independent of each other except for ones along the boundaries. Thus, in our GPU implementation we first sequentially compute the values of pixels on boundary seams between neighbouring videos; these seams are only one pixel wide so this can be done quickly. Given these boundary values, the other pixel values can be rapidly computed in parallel.

**Multi-Spline Blending:** This is the fastest Poisson-based blending approach as only a small linear system needs to be solved. It uses sparsely sampled tensor-product splines to approximate the smooth offset map (as in MVC blending) thus significantly reducing the size of the linear system to be solved. In our implementation we follow the parameter settings in [6] and use a spline spacing of 32. Since the majority of the computational cost is incurred by the equation solving step, the main difference between the CPU and GPU versions concerns how the linear system is solved.

### 5.1 Complexity

We analyze the complexity of the 4 representative algorithms. Since it only requires a linear combination at each pixel, the complexity of feather blending is $O(n)$ where $n$ is the number of pixels. For multi-band blending, since the scaling factor between neighbourhood levels is 0.5, the total number of pixels is increased by a fraction $\frac{4}{3}$ times and the complexity is still $O(n)$. With target region triangulation and adaptive boundary sampling, the cost for evaluating the membrane is $O(m)$ where $m$ is the number of pixels along the boundary, which will typically be $O(\sqrt{n})$. Interpolating the membrane values to all $n$ pixels again gives a total cost for MVC blending of $O(n)$. For multi-spline blending, we need to solve a linear system with $n/s^2$ unknowns where $s$ is the sampling space of the spline.

### 6 Experiments

Our experiments were performed on a PC with an Intel(R) Xeon E5-2620 2.0GHz CPU with 32GB memory, and an Nvidia GTX 970 GPU with 4GB memory; the
bandwidth between PC memory and the GPU memory was 4GB/s. The whole stitching pipeline as well as the blending algorithms were implemented in C++, while GPU implementations used CUDA.

6.1 Efficiency evaluation

In our experiments, we determine the time required by each algorithm as well as the amount of memory (CPU or GPU) it uses, for different video resolutions. The original resolution of each input video was 1920×1440. As GPU memory was limited, we down-sampled the input videos to resolutions of 1440×1080, 1024×768 and 800×600; the resolutions of the output blended videos were 2000×1000, 1422×711, and 1110×555 respectively.

Note that the time and memory costs only depend on the resolution of the input videos and the shape of the mask, and not on the content of the video. We thus just used one scene for this experiment. I/O time as well as pre-computing times were not considered, as we are interested in how suitable each method is for real-time needs. For each algorithm, we give below the memory cost for the CPU implementation, the memory cost and data transfer time for the GPU implementation, and computation time.

From the results in Tables 1–3, we can see that, given a GPU with sufficient memory, the multi-band and MVC blending approaches can both achieve real-time performance. Although we did not implement the GPU version of feather blending, it is fully parallelizable and should be faster than MVC. Thus, the GPU versions of these three algorithms can achieve real-time panoramic video stitching. Note that the memory cost of multi-band blending does not change when the video resolution changes. This is because before the construction of the pyramid, images were enlarged to a size which is a power of 2. For the CPU version of multi-band blending, we used the OpenCV implementation without further optimization.

| Algorithm | PC Memory | GPU Memory | Transfer Time | Computation Time |
|-----------|-----------|------------|---------------|-----------------|
| Feather   | 880 MB    | -          | -             | 280 ms          |
| MS        | 162 MB    | -          | -             | 2400 ms         |
| MB(CPU)   | 2100 MB   | -          | -             | 1200 ms         |
| MB(GPU)   | -         | 1200 MB    | 65 ms         | 10 ms           |
| MVC(CPU)  | 2300 MB   | -          | -             | 660 ms          |
| MVC(GPU)  | -         | 900 MB     | 60 ms         | <10 ms          |

6.2 Visual quality evaluation

Visual quality of videos can be evaluated objectively and subjectively. During the past few decades various video evaluation metrics have been proposed. How-
Table 2: Computation times and memory requirements of the four algorithms at 1422×711 resolution.

|                | PC Memory | GPU Memory | Transfer Time | Computation Time |
|----------------|-----------|------------|---------------|-----------------|
| Feather        | 720 MB    | -          | -             | 170 ms          |
| MS             | 82 MB     | -          | -             | 700 ms          |
| MB(CPU)        | 2100 MB   | -          | -             | 1200 ms         |
| MB(GPU)        | -         | 1100 MB    | 65 ms         | 10 ms           |
| MVC(CPU)       | 2100 MB   | -          | -             | 320 ms          |
| MVC(GPU)       | -         | 490 MB     | 30 ms         | <10 ms          |

Table 3: Computation times and memory requirements of the four algorithms at 1110×555 resolution.

|                | PC Memory | GPU Memory | Transfer Time | Computation Time |
|----------------|-----------|------------|---------------|-----------------|
| Feather        | 700 MB    | -          | -             | 120 ms          |
| MS             | 50 MB     | -          | -             | 360 ms          |
| MB(CPU)        | 2000 MB   | -          | -             | 1200 ms         |
| MB(GPU)        | -         | 1100 MB    | 65 ms         | 10 ms           |
| MVC(CPU)       | 1900 MB   | -          | -             | 220 ms          |
| MVC(GPU)       | -         | 330 MB     | 20 ms         | <10 ms          |

ever, objective evaluation is not that suitable for our specific application. The main reason is that blending artefacts such as bleeding and color inconsistency are hard to quantify. Instead, we chose to evaluate the visual quality of the blended video results subjectively.

We give below the scores of an evaluation provided by participants of the blending results of some typical frames. The blended videos are presented in the supplemental material.

### 6.2.1 Experiment setting and environment

Following[10], we conducted our experiment in a standard office environment and used integers from 1 and 5 inclusive for subjective scoring, where a higher score means higher visual quality. For each scene, we picked a 10 second segment of the output from each of the 4 algorithms, and an unblended result as a baseline. To test blending with obvious misalignments, we also gave used another scene for which the remapping was poorly done, resulting in obvious misalignment. The total length of video for each participant to view was 500 seconds ((4 + 2 + 3 + 1)×5×10). Each evaluation thus took less than 15 minutes. The participants in this experiment were 20 students, 10 of whom are majoring in computer vision and computer graphics, and the others are from other areas of computer science.
6.2.2 Experimental procedure

Each participant was first given training in which they were shown several typical scenes and their blending results. They were also provided with remarks by an expert in image processing, such as “there is an obvious seam and the color not very consistent near the seam” or “the moving object appears to be flickering”. Several kinds of artefact were also described. After this training session the participants were more aware of the visual quality of blended results.

6.2.3 Experiment results and analysis

The subjects were then shown the videos and asked to score them. We performed a filtering step [11] to reject scores considered to be outliers: considering all scores for a particular scene and a particular blending algorithm, a score was rejected if it was beyond two standard deviations from the mean score. An individual was regarded as an outlier if 1/3 of his scores were outliers. Finally only one participant was rejected.

After data filtering, we calculated the mean score over all 19 participants, for each algorithm, for each of 10 different scenes. We then calculated the mean and variance for each algorithm over 10 scenes, to determine the average performance and stability of each algorithm. The results are presented in Table 4. The results show that multi-band blending have the best stability and performance. Surprisingly, MVC blending generated worse results than no blending. This can be explained by the high variance assigned to no blending, as it varies widely in performance between scenes with obvious lighting differences and scenes with subtle lighting differences.

We give 3 typical frames of the blended video results in Figures 3–5 for the various approaches. An outdoor scene is shown in Figure 3. The result of feather blending has distinguishable boundaries in the sky, and ghosting exists near the telegraph pole. The multi-band blending result looks good at first glance, but compared with the MVC blending result, it is rather darker. The main problem in the MVC blending result is some color bleeding along misaligned boundaries. For multi-spline blending, as most pixel values are interpolated, the bleeding problem is not as severe as in MVC blending. An indoor scene is illustrated in Figure 4. Here, feather blending results in ghosting in the overlapping regions (e.g. the green leaves on the left). Multi-band blending works well for this scene. For MVC blending, because there are red lights along the boundary, red diffuses over a wide area. Multi-spline blending again results in fewer bleeding artefacts than MVC blending. A further typical scene is illustrated in Figure 5. Here, the feather blending result has obvious lighting discontinuity along the seams while the sky looks too dark in the multi-band blending result. Both MVC blending and multi-spline blending work well for this scene.
Table 4: Mean and variance of quality scores for each algorithm, over 10 scenes.

|               | No blending | Feather | Multi-band | MVC  | Multi-spline |
|---------------|-------------|---------|------------|------|--------------|
| Mean          | 2.39        | 3.43    | 3.61       | 2.17 | 2.93         |
| Variance      | 0.86        | 0.11    | 0.09       | 0.32 | 0.17         |

7 Conclusions

We have compared 4 representative blending algorithms as well as two GPU versions of them. We have captured various kinds of scenes, and used them to evaluate the efficiency and effectiveness of these algorithms. We make the following observations:

- For 2K resolution output video, the GPU version of MVC blending and GPU version of multi-band blending are fast enough for realtime use (assuming sufficient bandwidth between PC memory and GPU memory). Since feather blending is fully parallelizable and has the least computation, it is easy to implement for realtime use.

- Feather blending suffers from ghosting artefacts, but when it comes to video, this is not necessarily obvious. This simple method always works well.

- MVC blending provides the best visual quality when the source videos are well aligned, but this is rare in real world conditions. It suffers from severe bleeding artefacts when misalignment exists. In video, bleeding artefacts make the content flicker, and this significantly disrupts the viewing experience.

- Multi-spline blending suffers less from bleeding than MVC blending, but more so than with multi-band blending. Another problem is that this approach needs to solve a linear equation, which is time consuming.

- Multi-band blending can handle various kinds of scenes, and its GPU version is efficient. Despite it being an early method, it provides a good balance between computational cost and visual quality, and is suitable for realtime panoramic video stitching.

In future, we plan to capture more kinds of scenes for testing. We have made our code and videos publicly available, and hope our work can stimulate more researchers to develop more efficient video blending algorithms.
Figure 3: Blended video frame for an outdoor scene. Top to bottom: no blending, feather blending, multi-band blending, MVC blending and multi-spline blending.
Figure 4: Blended video frame for an indoor scene. Top to bottom: no blending, feather blending, multi-band blending, MVC blending and multi-spline blending.
Figure 5: Blended video frame for another outdoor scene. Top to bottom: no blending, feather blending, multi-band blending, MVC blending and multi-spline blending.
References

[1] Szeliski, R.: Image alignment and stitching: A tutorial. Technical Report MSR-TR-2004-92, Microsoft Research (October 2004)

[2] Perez, P., Gangnet, M., Blake, A.: Poisson image editing. ACM Transactions on Graphics (SIGGRAPH’03) 22(3) (2003) 313–318

[3] Agarwala, A.: Efficient gradient-domain compositing using quadtrees. ACM Transactions on Graphics 26(3) (2007) 94

[4] Farbman, Z., Hoffer, G., Lipman, Y., Cohen-Or, D., Lischinski, D.: Coordinates for instant image cloning. ACM Transactions on Graphics 28(3) (2009) 67

[5] Burt, P.J., Adelson, E.H.: A multiresolution spline with application to image mosaics. ACM Transactions on Graphics 2(4) (1983) 217–236

[6] Szeliski, R., Uyttendaele, M., Steedly, D.: Fast poisson blending using multi-splines. In: International Conference on Computational Photography (ICCP 11), IEEE (April 2011)

[7] Jia, J., Sun, J., Tang, C.K., Shum, H.Y.: Drag-and-drop pasting. ACM Transactions on Graphics (2006)

[8] Tao, M.W., Johnson, M.K., Paris, S.: Error-tolerant image compositing. International Journal of Computer Vision 103(2) (2013) 178–189

[9] Brown, M., Lowe, D.G.: Automatic panoramic image stitching using invariant features. International Journal of Computer Vision 74(1) (2006) 59–73

[10] : Methodology for the subjective assessment of the quality of television pictures. ITU-R Recommendation BT.500-13. (January 2012)

[11] Liu, H., Klomp, N., Heynderickx, I.: A no-reference metric for perceived ringing artifacts in images. IEEE Transactions on Circuits and Systems for Video Technology 20(4) (2010) 529 – 539