Omnistereo video textures without ghosting

Vincent C.-Couture
Université de Montréal
chapdelv@iro.umontreal.ca

Michael S. Langer
McGill University
langer@cim.mcgill.ca

Sébastien Roy
Université de Montréal
roys@iro.umontreal.ca

Abstract

An omnistereo pair of images provides depth information from stereo up to 360 degrees around a central observer. A method for synthesizing omnistereo video textures was recently introduced which was based on blending of overlapping stereo videos that were filmed several seconds apart. While it produced loopable omnistereo videos that can be displayed up to 360 degrees around a viewer, ghosting was visible within blended overlaps. This paper presents a stereo stitching method to render these overlaps without any ghosting. The stitching method uses a graph-cut minimization. We show results for scenes with differents types of motion, such as water flows, leaves in the wind and moving people.

1. Introduction

Traditional stereo imaging uses two cameras converging at some depth to capture two videos of a scene from slightly different viewpoints. The stereo pair can then be displayed using a stereo projector or monitor and, using stereo glasses or an autostereoscopic display, users can fuse the stereo images and enhance their perception of scene depth.

This paper addresses the problem of capturing omnistereo video. Here, the stereo video to be displayed should cover a very wide field of view – up to 360 degrees. Omnistereo video can be used in immersive environments to provide stereo cues all around an observer looking in an unknown direction, for instance, in CAVEs [10, 9, 19, 3, 4] for navigation in a virtual environment. Another potentially interesting application is to use existing consumer level desktop stereo displays or stereo TV. Allowing a user to actively pan over a 360 degree panorama could improve the user experience in applications such as Google Street View [2].

The standard approach for synthetizing omnistereo images of *static scenes* is to use a conventional stereo video rig where two cameras follow a circular motion and capture a space-time volume of images [14, 13, 19, 20]. From these two 3D space-time volumes, one extracts a few hundred small field of view stereo "slits" and stitches them together to produce a single larger field of view static stereo pair [1]. The stereo slits capture scene points on the median plane perpendicular to the baseline of the camera rig, and hence give maximum stereo disparity information [20].

As the stereo rig is rotated, each camera rotates but also translates. The translation introduces some parallax from frame to frame which produces visible seams when the images boundaries are stitched together. The advantage of using many small slits is that the slit-to-slit camera translation is small and so is the parallax. An alternative method for reducing parallax in stereo panoramas is to use large stereo frames, which are chosen such that the left edge of one frame and the right edge of the next frame lie on the line through the two camera positions [6, 8]. See Figure 1. This method eliminates horizontal stitching misalignments that are due to parallax at the frame edges, though vertical stitching misalignments can still occur at the top and bottom of the edges of the frames [8].

The method of [8] used large stereo frames to produce omnistereo video textures that are loopable in time [22]. The main limitation of that method is that it used simple blending to handle overlaps between frames. It was shown [7] that this blending method works fine for certain types of motions such as water waves, but that it produces noticable ghosting [11] for motions of well-defined visual features that can be tracked over time.

The contribution of the present paper is to improve the method of [8] by replacing the blending method with a graph cut based stitching method which finds an optimal seam between overlapping full frame stereo videos.

The paper is organized as follows. Section 2 briefly reviews previous work related to monocular panorama stitching of dynamic scenes. Section 2.1 summarizes the work presented in [8] which produces omnistereo videos using blending. Section 3 presents the new method to render overlapping regions, which uses a graph-cut minimization to find an optimal seam. Section 4 gives experimental results on a few scenes. We conclude in Section 5.

[1] Alternatively, Peleg *et al.* [20] showed how to extract stereo slits from a single space-time volume.

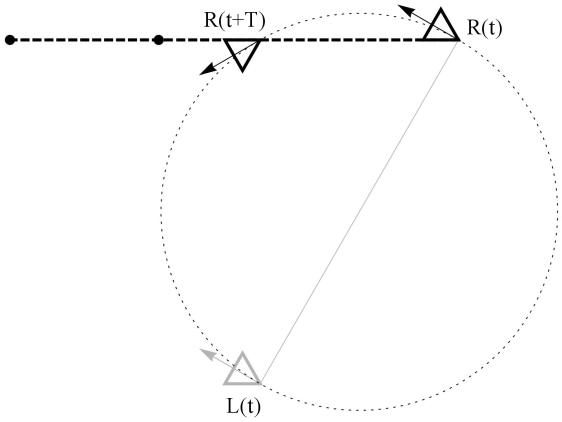

Figure 1. For a stereo rig with parallel cameras L and R, we consider two scene points (indicated by small black circles) that lie in the plane that contains the stereo camera motion and that enter and then exit the field of view of the right camera at times t and t+T respectively. Note that the points are colinear with these two camera positions.

2. Previous work

In addition to the slit-based method mentioned in Sec. 1, a related set of methods have been proposed for rendering monocular panoramas of captured dynamic scenes. These methods rearrange either slices [21] or small 3D blocks [1] from a monocular space-time volume, so as to avoid visual seams between the slices or blocks. An example is the *dynamosaicing* method [21] which uses graph cuts to compute a time evolving surface in a video's space-time volume and then makes a video mosaic by stitching the surfaces together. A second graph cut based approach [1] is *panoramic video textures*. This method renders a video seen by a rotating camera. Rather than selecting and stitching together slices in the space-time volume, it selects and stitches small space-time blocks. Such methods have been used successfully to generate panoramas from videos taken by a (purely) rotating camera.

To extend these block (or slit/slice) methods to stereo video, one needs to find corresponding blocks in two space-time volumes. The challenge is that the corresponding blocks must capture *the same scene points at the same time*, otherwise the temporal asynchrony of a moving point will produce incorrect stereo disparities and occlusion cues, namely when a scene point is occluded over some set of frames. Rather than using small blocks to compute stereo motion correspondence, it has been shown that one can use large space-time blocks, namely full frames [8]. The idea is that these full frames automatically capture correct stereo correspondence, except near the left and right edges of the frame. We summarize the method of [8] in the next section.

2.1. Omnistereo video texture using full frames

The problem formulation is as follows. A pair of videos is captured by a stereo rig rotating around a vertical axis. Each camera follows a circular path for a full 360 degrees. All frames are first registered to form a stereo XYT volume. This requires autocalibrating the two cameras[2]. Let the left/right volumes have N frames each, frame N is registered with frame 0. To simplify the explanation of the method, we assume camera rotation speed is constant as if it were controlled by a motorized tripod head. In a more general situation of a manually guided rotation (as in our case), the space time volume is "rectified" by temporally interpolating the original frames.

The left and right space-time volumes are each partitioned into blocks of T frames which are shifted in time to start at frame 0, which yields a larger field of view video. Figure 2 shows an example in which the entire stereo video is 120 seconds long and is partitioned into five blocks that are 24 seconds each. T is chosen so that the overlap width between blocks is a third of the original frame width. For a input stereo sequence of about 2 minutes, using such a T value yields between 10 and 12 block overlaps.

In [8], the frames within overlap regions were blended together using a simple linear ramp function. Such blending introduced ghosting of moving scene points within the left or right videos. While the disparity of each stereo block is correct, the blending of two overlapping stereo blocks creates ghosting [11] which is visually salient in many common cases. Examples are leaves and branches swaying in the wind [7] or people walking. The main motivation and contribution of the present paper is to present an alternative method for dealing with the overlap between neighboring stereo frames which avoids ghosting.

3. Avoiding ghosting using graph cuts

We present a method to find an optimal seam within the overlap region between neighboring frames and stitch the frames together at this seam. Rather than blending, we use a graph-cut method similar to [16] to find an optimal seam between the two blocks within an overlap. We first consider the monocular case in Sec. 3.1 and define the graph and the cut. We then describe how to adjust exposure differences between blocks in Sec. 3.2. This is necessary because the scene's dynamic range is typically large and the cameras' exposures are adjusted automatically as the rig is rotated. We extend the graph cut method to the stereo case in Sec. 3.3.

[2]Camera parameters include focal length, radial distortion as well as the position and orientation of the camera at each frame.

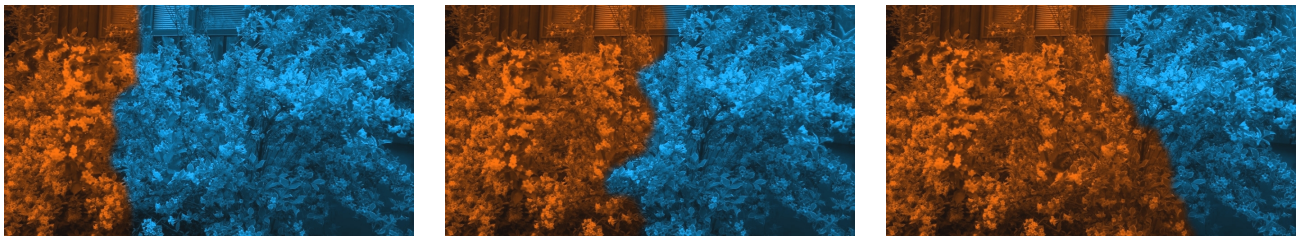

Figure 3. Evolution of the optimal seam between two blocks as the overlap moves from left to right over time.

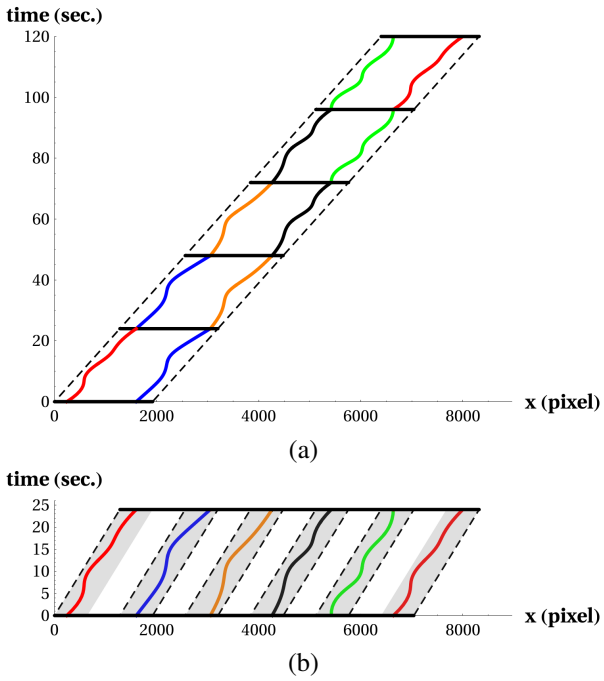

Figure 2. For a frame sequence captured by a camera performing a full turnaround in $N = 5T$ seconds at constant speed, the cut finds a low cost seam between the right border of a frame and the left border of frame T seconds later. (a) The continuous cut shown twice in the full original space-time volume divided in five non-overlapping blocks. (b) The same continuous cut shown with blocks aligned to start at the same time. The *overlap* regions are indicated in gray.

3.1. Monocular seam

In [16], a texture (image or video) is synthesized by stitching together several patches or space-times blocks. An optimization is required to find both the location and the shape of a patch. The problem we solve here is more constrained. The locations of all the video blocks and their overlapping regions are fixed, and the goal is to find an optimal seam between the two blocks in each overlap regions (see Fig. 2.)

This problem can solved using a graph-cut minimization [5] in the following way. We use a matching cost that measures pixel differences between the two blocks within the overlap [16]:

$$\mathcal{C}_{mono}(s, A, B) = \|A(s) - B(s)\| + \|A(s+1) - B(s+1)\| \quad (1)$$

where $A(\ )$ and $B(\ )$ are the pixel luminances in block A and B, respectively, and s and $s+1$ are two adjacent pixel positions in space or time. (An RGB distance could also be used.) The above cost function assumes brightness constancy of a pixel. This assumption typically does not hold for raw image intensities, since the camera automatically adjusts exposure during rotation. We handle this with a preprocessing step which is discussed in Sec. 3.2.

By definition, the graph cut finds a surface in XYT such that the sum of pixel differences across the surface is a minimum. All arcs connecting adjacent pixels s and $s+1$ are given flow capacity $\mathcal{C}_{mono}(s, A, B)$. Constraint arcs with infinite capacity are also added to ensure continuity with pixels outside the overlap, namely nodes to the left of block A in the overlap are linked to a source node, and nodes to the right of block B in the overlap are linked to a sink node.

While the results in [16] show that the optimal cut usually varies locally where motion is minimum, in our case the moving overlap *forces* the cut to move (see Fig. 3). Typically, the cut stays for many frames at roughly the same location, adjusting itself to motion in the two blocks. But as the overlap eventually shifts away from this location, the cut must jump to another location. The cost minimization reduces the chance of having a visible "jump" of the cut. An optimal jump is chosen such that the two frames have similar intensities. There is no guarantee, however, that there will exist a cut such that the positions of scene points in the two block videos can be perfectly aligned across the cut. To reduce these jump artifacts, we use a feathering kernel in both in space and time to blend the two blocks in a small neighborhood of the cut (10 pixels in each spatial direction and 10 frames in time), rather than directly stitching them together at the seam. This adds a small amount of ghosting around the cut, but much less than in [8] where a large blending window (width of 320 pixels) covering the whole overlap was used.

We solve for the cuts of neighboring blocks sequentially, adding constraint arcs from a node to the source or sink to enforce that the cut of the first frame of an overlap is the same as the cut of the last frame in the previous overlap.

(a) (b)

Figure 4. Optimal transition for a single frame within an overlap rendered (a) using the original intensities. (b) after a histogram equalization of the darker block of frames so that it matches the lighter block.

One could solve one global cut, and enforce that the cut ends at frame N at the same pixel locations it started at frame 0. But this would have been more memory intensive and would have reduced performance.

3.2. Exposure Adjustment

As stated above, the method assumes brightness constancy within overlaps, that is, $A(s)$ and $B(s)$ have the same luminance (or RGB) if they correspond to the same scene point. This assumption will typically break even in the case of static Lambertian surfaces if cameras automatically adjust exposure as the cameras rotate. Such exposure compensation is to be expected in real scenes, since the dynamic range varies considerable across a scene. Therefore, we needed[3] to make exposure adjustments before the graph cut is computed.

Suppose, for example, that the mean scene luminance in the window in block A were less than in the overlapping block B at some frame. Our videos are shot in shutter speed priority mode, and so the camera would adjust the f-number when capturing A by reducing it relative to B. As a result, scene points that are visible in the overlap of A and B would be be overexposed in A relative to B. We compensate for such differences in exposure by applying a histogram transfer function for each color channel which maps the histogram of the darker block B so that it has the same histogram as the lighter block A. The transfer function we use is based on the histogram matching method [12, 17]. Fig. 4(a) shows a frame using the original intensities on each side of the cut. Fig. 4(b) shows the adjusted intensities, namely the intensities in the right side (block B) have been increased. We found that we needed to adjust the intensities up rather than down because adjusting down caused partly saturated pixels to take on the wrong hue. We also needed to adjust the intensities outside the overlap region since otherwise we would have an intensity discontinuity between the raised region in the AB overlap and the non-raised region in B which is to the right of the AB overlap. We did so by applying an intensity histogram transfer in the non-overlapping region of B. Specifically, we decreased the magnitude of the histogram adjustment gradually from the edge of the AB overlap across B and up to the start of the next overlap region (BC), at which point the histogram adjustment is zero. Finally, if B was also underexposed relative to the next region C, then a convex combination of two histogram adjustments was needed in the non-overlap region in B, between A and C. The formula we used for the adjusted intensities at a pixel in the non-overlapping region in B in this case was

$$I_{new} = I_{original} + \alpha(\Delta I)_{AB} + (1-\alpha)(\Delta I)_{BC}$$

where $\alpha \in [0,1]$ is the fraction of the distance from the right edge of A to the left edge of C, and the ΔI are the intensity adjustments needed at that pixel to compensate for exposure differences as determined by the histogram transfer function. For the situation in which B has lower exposure than only one of A or C, just one of the terms containing α is used. If B has a greater exposure than both A and C, then no adjustment in the non-overlapping region in B is needed.

Note that this histogram adjustment is a pre-processing step. It is done before the graph cut is computed and so the cost function is defined by these adjusted RGB intensities. These adjusted intensities are used in the rendering as well.

[3] Note that one could avoid this problem by using high dynamic range video cameras [15] and tone mapping tricks, but we wished to see what could be achieved with consumer level cameras.

3.3. Stereo seams and window violations

We now turn to the problem of computing the graph cut for the stereo pair. Rather than computing the cuts in the left and right panoramas independently using the scheme outlined in Sec. 3.1, we incorporate the *disparity d of the cut* as part of a single graph cut minimization. For this we define a *stereo cost* at each edge in the graph:

$$\begin{aligned} \mathcal{C}_{stereo}(s, d, A, B) &= \mathcal{C}_{mono}(s + d, A_l, B_l) \\ &+ \mathcal{C}_{mono}(s, A_r, B_r) \end{aligned} \quad (2)$$

where s is a pixel position in a panorama and A_l, B_l and A_r, B_r are overlapping video blocks in the left and right camera, respectively, and disparity d is the difference in x position of the cut in the left vs. right frames at any seam point.

Several versions of this scheme are possible. A simple one, which we use, is to fix d in advance to be approximately the maximum disparity in the scene. This matches the disparity of the cut to that of closest objects, which typically draw more attention from viewers since they are in the foreground. Alternative schemes which are more complicated include solving for d as part of the graph cut minimization, or trying to estimate d at each pixel and each frame using a stereo-motion algorithm and then imposing the computed values of d in the cost function.

Note that when the disparity d of the cut is different from that of some scene point, there will be frames in which that scene point is visible to one camera but not the other. In standard 3D cinematography, this situation is not a problem as long as the depth of the scene point is greater than the depth of the stereo window, i.e. avoiding the window violation problem [18]. However, for panoramic stereo videos with seams, the situation is more complicated because there is no stereo window behind which such points can be partly occluded. When such scene points are not moving, there is no problem with having an "incorrect" disparity of the cut since the point's position within the left or right frame will not change as the cut passes over it, and so the cut will not be visible. However, if a point is moving then it may not be possible to find the correct cut even if one knows the disparity d of a scene point. The problem is that the seam stitches together points that are separated in time by T frames. For scene points that are moving, their image positions and disparities at time t and $t + T$ will in general not be the same.

The cost function that we use is not designed to solve this more general stereo-motion window violation problem. Rather, it is designed simply to find seam positions that minimize the intensity differences for the chosen disparity d. This solution produces cuts that are typically not visible in practice.

4. Experiments

Stereo videos were captured using two Canon HFS11 cameras on a fixed tripod with a rotating head. The distance between the centers of both lens was 6.5 cm, similar to the typical distance between human eyes. The field of view was 55 degrees. To synchronize frame capture and zoom, both cameras were controlled through the LANC protocol.

All computations were performed on a laptop (Intel dual core T7500 processor, 2GB of RAM). To speed up performance, we down-sampled the HD original content from 1920×1080 resolution to 960×540. The resolution of the final videos was about 7000×540 pixels. Each overlap region covered 320×540 pixels (i.e. a third of a full frame). The graph cut between neighboring blocks was solved at quarter resolution, i.e. 80×135 pixels. Each block was typically 200 to 300 frames. Solving the cut for each block took about 30 seconds of CPU time.

We present three scenes, namely `Boats`, `River` and `Park`. See Fig. 5 for a third of a single frame of each of the resulting panoramas, presented as red/cyan anaglyphs. The reason we show a third of a frame only is that the 12:1 aspect ratio (horizontal:vertical) of the entire frame is very large and the vertical dimension would be excessively squeezed if we presented the entire panorama.

The motion in the scenes is mostly texture-like, *e.g.* water flow and leaves in the wind. While it was shown in [7] that blending already gives good results for water, our graph-cut technique clearly creates less ghosting for leaves and other vegetation for all scenes we have tried. For example, Fig. 6(a,b) compares single frames for a video containing foliage blowing in the wind. In the blended case (a) which uses the technique of [8], ghosting of the leaves is severe. The graph cut case (b) has no ghosting visible.

The scenes also contain various examples of non-texture like motions, such as people playing cards, walking or riding a bicycle. Fig. 6(c,d) shows that the moving men are ghosted in (c) but not in (d). In this example, the graph cut finds a seam where the positions of the people are roughly aligned in the two images. While the graph-cut technique works well when people remain near the same area, it is not well suited when objects or people undergo significant position change over time, such as a moving car or a person walking across the scene. This is the main difference between video texture methods and general video methods. Nonetheless, we observe that the cut gracefully handles some cases in the presence of natural occluders. For example, if someone is walking and happens to be occluded by a tree when in an overlap, then the cut makes the person disappear as he/she will remain hidden behind the tree. See `River` sequence in the supplemental material. However, if there is no such natural occluder, then the person disappears behind an "invisible" occluder defined by the cut. An example can be found in the `Park` sequence.

Figure 5. Third a frame (120^o field of view out of the full 360^o) in the omnistereo videos of our three result scenes, namely (a) `Boats` (b) `River` and (c) `Park`. For the reader's convenience, left/right frames have been converted to grayscale and are presented as red/cyan anaglyph.

(a) blending (b) graph cut (c) blending (d) graph cut

Figure 6. Comparison of a single frame in an overlap of a video containing (a,b) foliage blowing in the wind or (c,d) moving people. In both examples, ghosting is reduced considerably when using thin feathering around an optimal cut (b,d) compared to using large blending as in [8] (a,c).

5. Conclusion

We have presented an improved method for computing panoramic stereo video textures using a pair of off-the-shelf consumer video cameras. The method uses the full frame video which gives automatic stereo motion synchronization for the vast majority of the visible points. The main contribution is to improve the method of [8] which used blending to combine overlapping regions, and which often resulted in ghosting for regions containing motion. Instead of blending, our method computes an optimal transition – specifically, a graph cut – in the overlap region between neighboring blocks. This transition was shown to work better for certain texture like motions such as foliage, which create ghosting when blended.

References

[1] A. Agarwala, K. C. Zheng, C. Pal, M. Agrawala, M. Cohen, B. Curless, D. Salesin, and R. Szeliski. Panoramic video textures. *ACM Transactions on Graphics*, 24(3):821–827, 2005. 2

[2] D. Anguelov, C. Dulong, D. Filip, C. Frueh, S. Lafon, R. Lyon, A. Ogale, L. Vincent, and J. Weaver. Google Street View: Capturing the world at street level. *Computer*, 43(6):32–38, June 2010. 1

[3] P. Bourke. Synthetic stereoscopic panoramic images. *Lecture Notes in Computer Science (VSMM 2006)*, 4270:147–155, 2006. 1

[4] P. Bourke. Omni-directional stereoscopic fisheye images for immersive hemispherical dome environments. *Computer Games and Allied Technology*, pages 136–143, May 2009. 1

[5] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23, 2001. 3

[6] V. Couture, M. S. Langer, and S. Roy. Capturing non-periodic omnistereo motions. In *10th Workshop on Omnidirectional Vision, Camera Networks and Non-classical Cameras (OMNIVIS)*, Zaragoza, Spain, 2010. 1

[7] V. Couture, M. S. Langer, and S. Roy. Perception of blending in stereo motion panoramas. *ACM Transactions on Applied Perception*, 9(3), 2012. 1, 2, 5

[8] V. Couture, M. S. Langer, and S. Roy. Panoramic stereo video textures. *IEEE International Conference on Computer Vision (ICCV)*, Nov. 2011. 1, 2, 3, 5, 6, 7

[9] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti. Surround-screen projection-based virtual reality: the design and implementation of the cave. In *ACM SIGGRAPH Proceedings*, pages 135–142, New York, NY, USA, 1993. 1

[10] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon, and J. C. Hart. The cave: audio visual experience automatic virtual environment. *Communications of the ACM*, 35(6):64–72, 1992. 1

[11] J. Davis. Mosaics of scenes with moving objects. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 354 –360, 1998. 1, 2

[12] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Addison-Wesley Publishing Company, 1992. pp. 173-182. 4

[13] H.-C. Huang and Y.-P. Hung. Panoramic stereo imaging system with automatic disparity warping and seaming. *Graphical Models and Image Processing*, 60(3):196–208, 1998. 1

[14] H. Ishiguro, M. Yamamoto, and S. Tsuji. Omni-directional stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):257–262, 1992. 1

[15] S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski. High dynamic range video. *ACM Trans. Graph.*, 22(3):319–325, July 2003. 4

[16] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics*, 22(3):277–286, July 2003. 2, 3

[17] J. S. Lim. *Two-Dimensional Signal and Image Processing*. Prentice-Hall, 1990. pp. 453-459. 4

[18] B. Mendiburu. *3D Movie Making: Stereoscopic Digital Cinema from Script to Screen*. Focal Press, 2009. 5

[19] T. Naemura, M. Kaneko, and H. Harashima. Multi-user immersive stereo. *IEEE International Conference on Image Processing*, 1:903, 1998. 1

[20] S. Peleg, M. Ben-Ezra, and Y. Pritch. Omnistereo: Panoramic stereo imaging. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(3):279–290, 2001. 1

[21] A. Rav-Acha, Y. Pritch, D. Lischinski, and S. Peleg. Dynamosaicing: Mosaicing of dynamic scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(10):1789–1801, 2007. 2

[22] A. Schödl, R. Szeliski, D. H. Salesin, and I. Essa. Video textures. In *ACM SIGGRAPH Proceedings*, pages 489–498, New York, NY, USA, 2000. 1

[23] H. Woeste. *Mastering Digital Panoramic Photography*. Rocky Nook, 2009. 1