Animated Panorama from a Panning Video Sequence

Fay Huang[1], Kun-Ming Yang[1], Zhang-Jun Wei[1], Augustine Tsai[2], and Jui-Yang Tsai[2]

[1]Institute of Computer Science and Information Engineering
National Ilan University, Yi-Lan, 26047 Taiwan, R.O.C.
[2]Emerging Smart Technology Institute
Institute for Information Industry, Taipei, 105 Taiwan, R.O.C.
Email: fay@niu.edu.tw*

Abstract

In the recent applications of virtual reality and augmented reality, panoramic-image-based representation of real world environments has received much attention for its advantages over the traditional 3D modeling approaches, such as shorter generation times, faster rendering speeds, higher photorealism, and less storage needed. The drawback is however that it stores only static information of the scene. Panoramic video has been proposed to preserve wide field-of-view dynamic characteristics of the natural environment, which is especially suitable for outdoor sceneries. Its limitation is lack of control of such video. In this paper, a newly developed software program is presented, which takes a single video sequence from a panning camera, and generates a cylindrical panorama embedded with video textures, called animated panorama. The program has the following functions: seamless video textures generation, undesired object removal by video inpainting, and animated panorama player. We claim that the quality of the animated panorama produced by our program is satisfactory to virtual reality applications and the computation time is acceptable for practical use.

Keywords: Panoramic image, video textures, video inpainting, virtual reality

1 Introduction

Panoramic images have been widely used in various virtual reality applications to achieve realistic rendering in real-time. In an abstract geometric sense, captured data are mapped onto a surface, which may be a sphere, cube, or cylinder, and thus called spherical, cubic, or cylindrical panorama, respectively. A 360° cylindrical panoramic image can be generated by various techniques, such as mosaicing or stitching [2], or can also be acquired using specialized sensors such as catadioptric [9]. In particular, stitching is an efficient method if taking only camera panning motion into account. Therefore, panoramic image stitcher has becoming a standard built-in feature for many models of digital cameras. A common scenario would be that a user holds the digital camera on his/her hand as steady as possible or place the camera on a rotatable tripod, and captures a sequence of images during a rotating motion. The camera is supposed to rotate with respect to a vertical rotation axis possing through the camera's nodal point. Any pair of successive images should have at least 30 percents overlapping field of view. Many cameras' built-in image stitchers or commercially available panorama makers would automatically estimate all the parameters required for generating the panoramic image based on the provided set of sequential images. The underlying calculations include feature detection, matching, warping, and color blending.

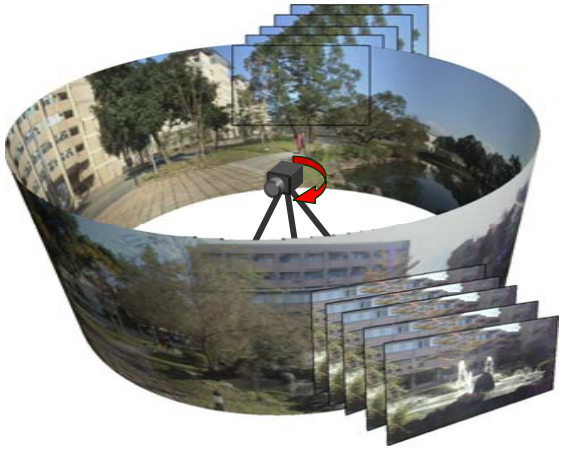

Figure 1: Our program accepts a video sequence captured by a panning camera as input and provides a player which allows users to freely navigate through the resulting animated panoramic image. The dynamic portions of the scene are represented by the embedding video textures.

*978-1-4244-9631-0/10/$26.00 ©2010 IEEE

Figure 2: An example of image stitching result. The input video consists of 1,676 frames, and the resulting panorama has resolution of 5925 times 720 pixels.

Generating a cylindrical panoramic image from a rotating camera is a relatively new method [5]. Ideally the camera should be supported by a special designed rotating platform and captures one single image column (i.e., 1D image) instead of a 2D image at each time instant. By this approach, it is possible to bypass images warping. Image warping is a process highly depends on the accuracy of the estimated camera parameters. The resulting panoramic images would suffer from "double-image" effects if incorrect warping has been performed. Our developed program adopts the advantages from both methods. First, it accepts a video sequence captured by a rotating handheld or tripod camera as input, thus no special rotating equipment is needed. Second, the final panoramic image is generated by combining side-by-side image columns from different video frames without image warping. Our program allows users to freely navigate through the virtual environment represented by the resulting panoramic image. The concept of cylindrical representation of the environment is illustrated in Fig. 1.

During the video capturing process, it is often unavoidable to have walking pedestrians or moving vehicles in the scene, which will then appear in the resulting panoramic image. The program provides a simple user interface, which allows users to specify the undesired object to be removed and then fills the hole with the available background scene by video inpainting technique. The concept of inpainting algorithm used in our program is similar to [3, 12].

In order to increase the realistic impression while navigating in a panoramic-image-based virtual reality platform, various hybrid image or video approaches have been proposed [4, 6, 7, 10]. The goal is to preserve and illustrate the moving objects in the scene caused by nature or man-made forces, such as waving trees, fountains or streams, to enhance the realism of the virtual environment. Video data often contains spatial and temporal redundancy, especially for the above-mentioned object motions, information is highly repetitive or quasi-repetitive. The concept of video textures [11] is to create an impression that a video can be played continuously and infinitely based on a given sample video. Panoramic video textures [1] inherit the same concept and allow to generate a wide field-of-view video texture. It employs graph-cut algorithms to generate the video textures, which is very time consuming.

We aim to achieve a comparable quality of animation effects within a practically acceptable time. We call it an animated panorama, that is a cylindrical panoramic image embedded with multiple video textures, to distinguish it from static panoramic images. A pure graph cut method usually suffers from the drawback of high computational cost especially when applying on the spatio-temporal textures. In this paper, a simplified version of graph cut has been proposed and implemented to greatly reduce the computation time.

The contributions of the developed program include: First, an automatic approach that can generate video textures within acceptable time for practical use. Second, the image regions containing undesired moving objects can be eliminated and replaced by background scene. Third, a virtual reality player has been developed in conjunction with the animated panorama maker, which is capable of playing such format of animated panorama. The paper is structured as followings. The video registration and image stitching methods used will be described in Section 2. The approach of video texture generation will then be explained in Section 3. A brief introduction of program user interface is given in Section 4 and followed by experimental results and conclusions.

2 Image Stitching and Undesired Object Removal

The goal is to generate a cylindrical animated panorama, which is embedded with user-specified video textures, from a finite sequence of video frames captured by a single panning camera. This image acquisition approach is considered as an approximation to the method of generating a cylindrical panoramic image from a rotating camera. When a special designed rotating camera is used to capture a panoramic image as described in [5], the rotational angular difference between any pair of two adjacent shots remains constant, which can be determined from the camera's intrinsic param-

eters. Hence, every captured image contributes same amount of image data (i.e., equal numbers of image columns) to the resulting panorama. However, if using a sequence of video frames to generate the panoramic image, the rotation speed varies during recording, and as a result, each frame should contribute different numbers of image columns to compose the final panorama. Let c_i denote the number of image columns contributed from the ith frame to generate the panorama. Further, let $\bar{c}$ denote the expected average number of image columns contributed from all frames. Ideally, $\bar{c} = \frac{1}{N}\sum_{i=1}^{N} c_i$, where N is the total number of frames the camera captures in a 360 degree rotation.

In order to achieve a non-blurring and undistorted panoramic image, it is highly recommended to rotate the camera as steady as possible in slow motion during recording. A tripod will be useful to minimize camera shake. From our experiences, the scan time for a 360 degree rotation between 70 to 100 seconds is the optimal speed to balance the image quality and the processing time. If the camera's intrinsic parameters are known, then it is possible to estimate $\bar{c}$ by the following equation:

$$\bar{c} = \frac{2f}{\tau}\tan\left(\frac{\pi}{N}\right),$$

where f and τ denote the focal length of the camera and the pixel size (assuming squared pixels), respectively. The value of $\bar{c}$ is to be rounded to the neatest integer for later calculations. For each frame, only partial image region is used to perform video registration. Let W denote the width of an image frame in pixels. Image region bounded by columns $\left(\frac{W}{2} - \bar{c}\right)$ through $\left(\frac{W}{2} + \bar{c}\right)$ in each frame are gathered by our program to perform calculations for video registration. If users have no information of camera intrinsic parameters, then by default, program would select image region from column $\lfloor\frac{3W}{8}\rfloor$ to column $\lfloor\frac{5W}{8}\rfloor$.

The value of c_i for each $i \in \{1, 2, \ldots, N\}$ is determined by the result of video registration. There have been many algorithms developed for video registration for different purposes. Our problem is relatively simple due to the constrained camera motion. We define the similarity measure of two frames as the average pixel intensity difference within the overlapping region. Frames are registered based on the obtained minimum similarity value. For instance, if the ith frame and the $(i+1)$th frame have been determined having v_i columns overlapping, then $c_i = 2\bar{c} - v_i$. The resulting panoramic image is composed by stitching together c_i image columns from the ith frame, for all $i \in \{1, 2, \ldots, N\}$. A straightforward color blending scheme for each pixel of the ith frame has been implemented. The current pixel's color value is replaced by the average color value of three pixels

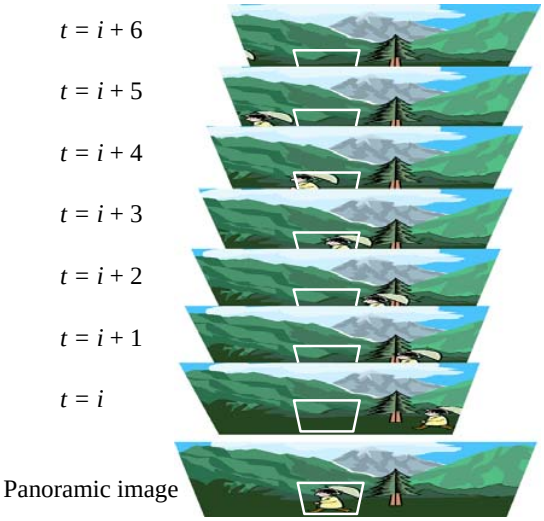

Figure 3: The concept of undesired foreground moving objects removal.

at the same position in frames $(i-1)$, i, and $(i+1)$. An example of image stitching result is illustrated in Fig. 2.

An image refinement feature, that is the ability to remove the undesired foreground moving objects, has been implemented. This makes our developed program more advance than other existing panoramic image makers or stitchers, such as Panorama Maker, PixMaker, Panorama Factory, AutoPano, and PTGui. In case that there were some moving objects in the scene during video recording, and users would like to remove them from the resulting panoramic image. The program provides an interface which allows users to specify regions to be refined which contain foreground objects, and the specified regions are replaced with background scene by exemplar-based image inpainting technique.

Figure 3 shows the concept of foreground moving objects removal, where the undesired object in the panoramic image (at the bottom) has been enclosed by a rectangle. The available source image frames from $t = i$ to $t = i + 6$ are displayed above the panoramic image. Our program automatically looks for a region among the source image frames that does not contain the foreground objects, such as the ith, $(i + 1)$th, $(i + 2)$th, or $(i + 5)$th frame and so on. The specified region in the panoramic image will then be replaced by the obtained region from one of the source image frames. It is naturally that the brightness of the original panoramic image and the replacement batch are different due to that they were acquired at different time instant. The program processes the brightness correction and linear blending to reduce the rectangular artifact. Figure 4 illustrates an example where the moving

pedestrians have been removed. The process can be done repeatedly until the refinement result is satisfactory.

3 Video Texture Generation

Texture synthesis is a process of algorithmically constructing a larger image based on a (small) sample image in a way that the resulting image preserves structural appearance of the sample image. Graph cut technique has demonstrated quite a few satisfactory texture synthesis results [8]. Video texture synthesis is a similar process but instead of increasing image size of each frame, the task is to extend the length of a sample video clip [1, 11]. Graph cut technique can easily be extended to 3D case. However a pure graph cut method is very time consuming especially when applying on 3D data. Our approach is somewhat a simplified version of graph cut method, but we claim that the quality of the video textures generated by our program is comparable to results of a pure graph cut method and the computation time can be greatly reduced.

Our goal is, given an image sequence of some moving objects, produce a video texture (often a shorter sequence) such that if playback this video texture repeatedly then the objects would appear to move continuously and infinitely. The width and height of the input sequence is determined by mouse drag and the length of the sequence depends on the camera panning speed. Our approach is to first overlap the starting and ending portions of the input image

Figure 4: An example of inpainting result. (a) is the original resulting panoramic image, (b) is an intermediate result after the region has been replaced, (c) is the result after brightness correction, and (d) is the result after blending.

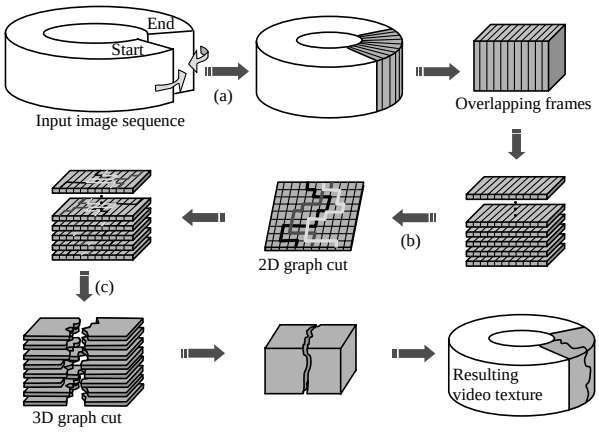

Figure 5: The procedures of video texture generation. The whole pipeline consists of three major tasks: (a) overlapping region determination, (b) minimum cut calculation, and (c) minimum cut approximation of 3D graph.

sequence, and then obtain a cutting surface within the overlap region. After performing this cut, the color difference between the neighborhood-pixels (i.e., pixels in the starting and ending portions of video frames respectively) along the cut-surface should be a minimum. Thus, the starting and ending portions of the image sequence can be stitched together seamlessly. We convert this problem into a finding minimum cut problem in a 3D graph, but instead of finding an absolute minimum cut, we intend to look for an approximated solution.

In graph theory, a cut is a partition of the vertices of a graph into two sets. Any edge crossing the cut is a cut edge. In the case of a weighted graph, a minimum cut following some specified cutting criteria is the cut that minimizes the sum of the cut-edge weights. The procedures and concepts of our video texture generation approach is illustrated in Fig. 5. The whole pipeline consists of three major tasks: (a) determination of overlapping region, (b) finding minimum cuts for each horizontal slice, and (c) finding an approximated minimum cut of the 3D graph (i.e., video data) based on the results from (b). In this section, we explain how the overlapping region is determined and how a simplified graph cut algorithm is applied to seamlessly stitch firstly two still 2D images and then secondly two videos sequences. Our goal is to generate seamlessly-looping video textures.

3.1 Overlapping Region Determination

The input image sequence is generally as short as less than 150 frames due to the panning camera motion. The length of the available input sequence is roughly inversely proportional to both the area of the specified animated region and the speed of the camera rotation. Sufficient image overlapping

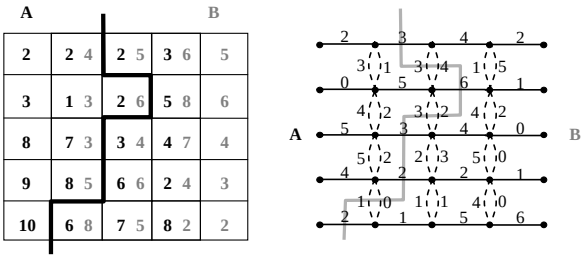

Figure 6: Left: a simplified example of two overlapping images A and B with three-pixel width overlapping region, where numbers denote the color intensities. Right: the constructed weighted graph of the overlapping region. The bold grey poly-lines indicate one of the possible cutting path for this example.

of the starting and ending portions of the video sequence is critical to the quality of stitching result, but too much overlapping region would greatly slow down the optimum cut-surface search. Our goal to minimize the overlapping region much as possible, and yet the region potentially contains the minimum cut.

The average intensity difference of all the image pixels in the overlapping region is calculated for various overlapping widths. Let $\bar{d}_k$ denote the average intensity difference, where k indicates the width (i.e., the number of image frames) of the overlapping region. If the input image sequence has T frames, then by default, the program tests various overlapping widths starting from $\lfloor \frac{T}{4} \rfloor$ to $\lfloor \frac{3T}{4} \rfloor$. The optimal overlapping width is defined by the minimum average intensity difference between the starting and ending portions of the video. However, user may also specify the minimum number of frames M of the resulting video texture. This is to ensure the long periodic motion characterizing the object's movement will be kept in the resultant video texture. In summary, our task is to find an optimal overlapping width v, i.e.,

$$v = \arg\min_k \{\bar{d}_k : k = (1..\lfloor \frac{T-M}{2} \rfloor)\}$$

3.2 Seamlessly 2D Stitching

In the next step, the program looks for a set of best seamless stitching paths for each horizontal slice of the determined overlapping region. The concept is illustrated in Fig. 5(b). The task is equivalent to finding minimum cuts in a converted 2D weighted graph. Figure 6 illustrates a small and grey scale example, where black and grey numbers stand for the image intensity values of image A and B, respectively. In this example two images A and B are to be stitched side by side, the goal is to find a cut-path from the top of the images to the bottom. After performing this cut, the color

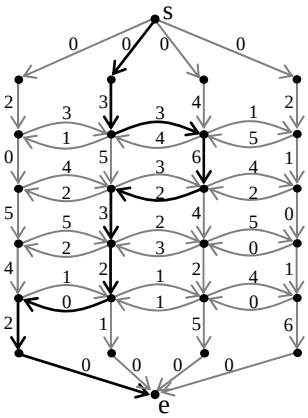

Figure 7: A directed weighted graph for the same example of Fig.6.

difference between the neighborhood-pixels (i.e., pixels in image A and pixels in image B) along the cut-path should be a minimum. In other words, it is to look for a least noticeable seam within the overlapping areas. We construct a weighted graph for the overlapping area where the nodes are the image pixels and the weight of each edge represents the color differences between two nodes. The cost of a cut is defined as the sum of all the edge weights in the cut. However, one thing to note is that only one of the two dashed edges at each pair of vertices would be taken into account depending on the direction of the cut path. The bold grey poly-lines indicate one possible cutting path, where the cost can be calculated as $(3+3+6+2+3+2+0+2) = 21$.

In order to visualize the 'implement' concept of finding the minimum cut of a graph for our case, we may transform the graph to a directed weighted graph as shown in Fig. 7. The task is then equivalent to finding a minimum-weight path from vertex s to vertex e. Viterbi algorithm was implemented to efficiently find the path of minimum weight in such a weighted graph. In order to extend the graph cut concept to video (i.e., a 3D data set), the program not only stores the minimum-weight path but also other paths having relatively smaller weights. Based on our experimental experiences, it is practical and sufficient to stores one minimum-weight path for each possible starting option. If two images have v-pixel width overlapping, then there are $v+1$ possible starting options, and hence $v+1$ paths will be stored for later calculations. In our simple example, since the overlapping width equals to three pixels, four best seamless stitching paths will be stored.

3.3 Seamlessly 3D Stitching

The determined overlapping region is shaded in grey in Fig. 5. We may consider it a gridded volume data. Each grid point contains two intensity

values, referring to image pixels in the starting and ending portions of the video accordingly. We use (x, y, t) to indicate a pixel location in a video, where t stands for the tth frame. The coordinate system is depicted in Fig. 8(left). For each horizontal slice (i.e., for each y) of this volume data, the program perform the 2D stitching algorithm described in Subsection 3.2. Multiple cut paths having relatively small costs were identified and stored for consideration.

The final cut-surface is the union of the subset of cut paths obtained at each horizontal slice. The task at this stage is to choose a best cut path at each slice such that all together they form a good approximation of the min-cut surface. Again, we may transform this problem into finding a minimum-weight path in a weighted simple graph. Let each vertex denote a cut path in one of the slice. Each pair of successive slices form a bipartite graph, which is a subgraph of the entire weighted simple graph to be processed. The weight of an edge is defined as the sum of intensity differences of all the pixels enclosed by these two cut paths between two successive horizontal slices. Figure 8(right) the region enclosed by two cut paths is shown as dashed area, where the black path belongs to the upper slice and the grey path belongs to slice beneath it. Viterbi algorithm is performed here to find the minimum-weight path and hence the optimal cut-surface to stitch the starting and ending portions of the video sequence.

Figure 9(left) illustrates one image frame from the resulting video texture of water fountain example. Since that the major movement direction of water in this case is vertical, the tidy horizontal strip artifact caused by processing horizontal slices before finding optimal 3D graph cut becomes more obviously visible. An alternatively approach to reduce such artifact is to process vertical slices in step Fig. 5(b) instead. That means, instead of processing xt-plane for each y, as shown in Fig. 8(left),

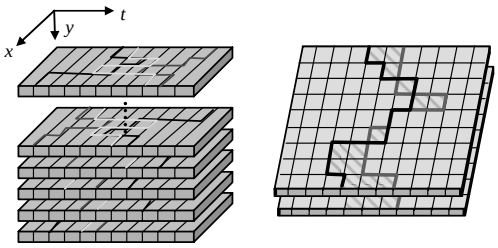

Figure 8: (Left) The coordinate system used to identify a pixel in the overlapping region. (Right) A cut path in each slice is represented by a vertex in a weighted simple graph. The weight of an edge connecting two vertices is defined as the sum of intensity differences of all the pixels enclosed by the associated cut paths between two successive slices.

Horizontal slice 2D graph cut Vertical slice 2D graph cut

Figure 9: Image frame from the resulting video texture of water fountain example. Left: result of horizontal slice 2D graph cut approach, where the tidy horizontal strip artifact is enclosed by circles. Right: result of vertical slice 2D graph cut approach, where the strip artifact is less visible.

program first processes yt-plane for each x and then obtains an optimal 3D graph cut within such overlapping region. The result obtained by this alternative approach is illustrated in Fig. 9(right), where the strip artifact is less visible.

4 Program User Interface

The developed program accepts MPG files as input. The graphical user interface of our program is shown in Fig. 10. After video registration and panoramic image stitching, users are able to view different portions of the resulting panoramic image by mouse drag. While navigating the panorama, users are able to specify regions(s) on the panoramic image for purposes such as video texture generation and moving object removal. A new window will pop out displaying each of the enclosed regions. The generated video textures will be played continuously also in those pop-out windows. Users may decide whether to embed those video textures into the final animated panorama based on the visual result of each resulting video texture.

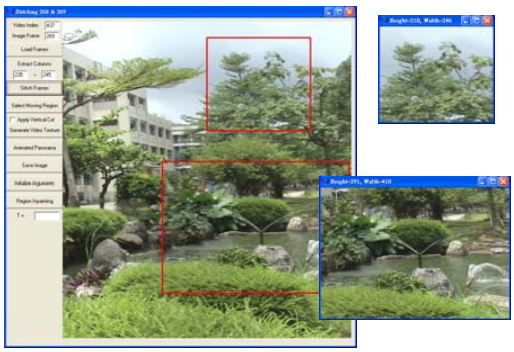

Figure 10: The graphical user interface of our program.

	Original sequence	Video texture	Image size	Computation time
Fountain1	115 frames	23 frames	247 x 362	45 seconds
Fountain2	105 frames	27 frames	382 x 209	36 seconds
Straws	80 frames	22 frames	180 x 210	9 seconds
Wave	283 frames	243 frames	612 x 292	112 seconds

Table 1: Information of the input and output videos.

5 Results

The program was written in Borland C++ Builder 6.0. The experiments were performed on Windows XP (Service Pack 3) operating system running on a Intel(R) Core(TM) i7 CPU 920 2.67 GHz with 3G of RAM. Sony DCR-SR62 digital video camera was used to capture the input video. The image resolution has been set to 480(width) $\times$ 720 (height) pixels with frame rate equal to 30 frames per second. The computation time for video registration and image stitching together is between 3 and 7 minutes depending on the number of image columns assigned to perform registration. This is considered acceptable if comparing to panoramic video texture, which would take hours to process.

Figure 11 shows four results of the generated video textures. The original fountain sequence extracted from the input video consists of 115 images of size 247 $\times$ 362 pixels. The resulting video texture has 23 frames. The video texture computation time for this example is 45 seconds. Every 4th frame of the resulting fountain texture sequence is shown in Fig. 11(left). Another three examples are another fountain, waving straws, and ocean wave shown in the second, third, and forth columns, respectively. Among them, fountain and straws were extracted from the same panorama video. The video information of all these examples and the computation times are summarized in Table 1. We aim to produce video textures consisting of small number of image frames unless user has specified a preferred minimum video texture length. For the wave example, since a complete periodic motion of the ocean wave takes about six seconds in this case, it is reasonable to constraint the length of the resulting video texture to be greater than 180 frames.

6 Conclusion

This paper presented a newly developed software program, which takes a single video sequence captured by panning the camera, and generates an animated panorama. The concept is to create a cylindrical panoramic image embedded with multiple video textures. The animated visual effects of our approach are comparable to results of panoramic video texture, which has been known a time-consuming task. The contributions of this work are as follows: (1) an approach of finding an approximated optimal cut-surface in a video sequence (i.e., a 3D space-time volume) has been proposed and implemented, thus the animated panorama can be generated within a practically acceptable time; (2) an image inpainting method has been developed which allows users to eliminate the undesired moving objects in the scene; and (3) a virtual reality player has been established which is capable of playing the resulting animated panorama.

7 Acknowledgment

This project is mainly supported by National Science Council (grand no. NSC 99-2221-E-197 -024) and partial experiments were sponsored by MOEA (ministry of economics affairs) project 98-ec-17-A-02-01-00809.

References

[1] A. Agarwala, C. Zheng, C. Pal, M. Agrawala, M. Cohen, B. Curless, D. Salesin and R. Szeliski. Panoramic Video Textures. In *Proc. SIGGRAPH 2005*, Los Angeles, USA, August, 2005, pages 821-827.

[2] S. E. Chen. QuickTimeVR - an image-based approach to virtual environment navigation. In *Proc. SIGGRAPH'95*, Los Angeles, California, USA, August, 1995, pages 29-38.

[3] A. Criminisi, P. Perez and K. Toyama. Region filling and object removal by exemplar-based image inpainting. In *IEEE Trans. Image Process.*, vol. 13, pages 1200-1212, 2004.

[4] A. Finkelstein, C.E. Jacobs and D.H. Salesin. Multiresolution video. In *Proc. SIGGRAPH 96*, New Orleans, Louisiana, USA, August, 1996, pages 281-290.

[5] F. Huang, R. Klette and K. Scheibe. *Panoramic Imaging: Sensor-Line Cameras and Laser Range-Finders*. Wiley, West Sussex, England, 2008.

[6] M. Irani and P. Anandan. Video indexing based on mosaic representation. In *Proc. IEEE 86*, 5, pages 905-921, 1998.

[7] D. Kimber, J. Foote and S. Lertsithichai. Flyabout: spatially indexed panoramic video. In *Proc. ACM MULTIMEDIA'01*, pages 339-347, 2001.

[8] V. Kwatra, A. Schodl, I. Essa, G. Turk and A. Bobick. Graphcut textures: image and

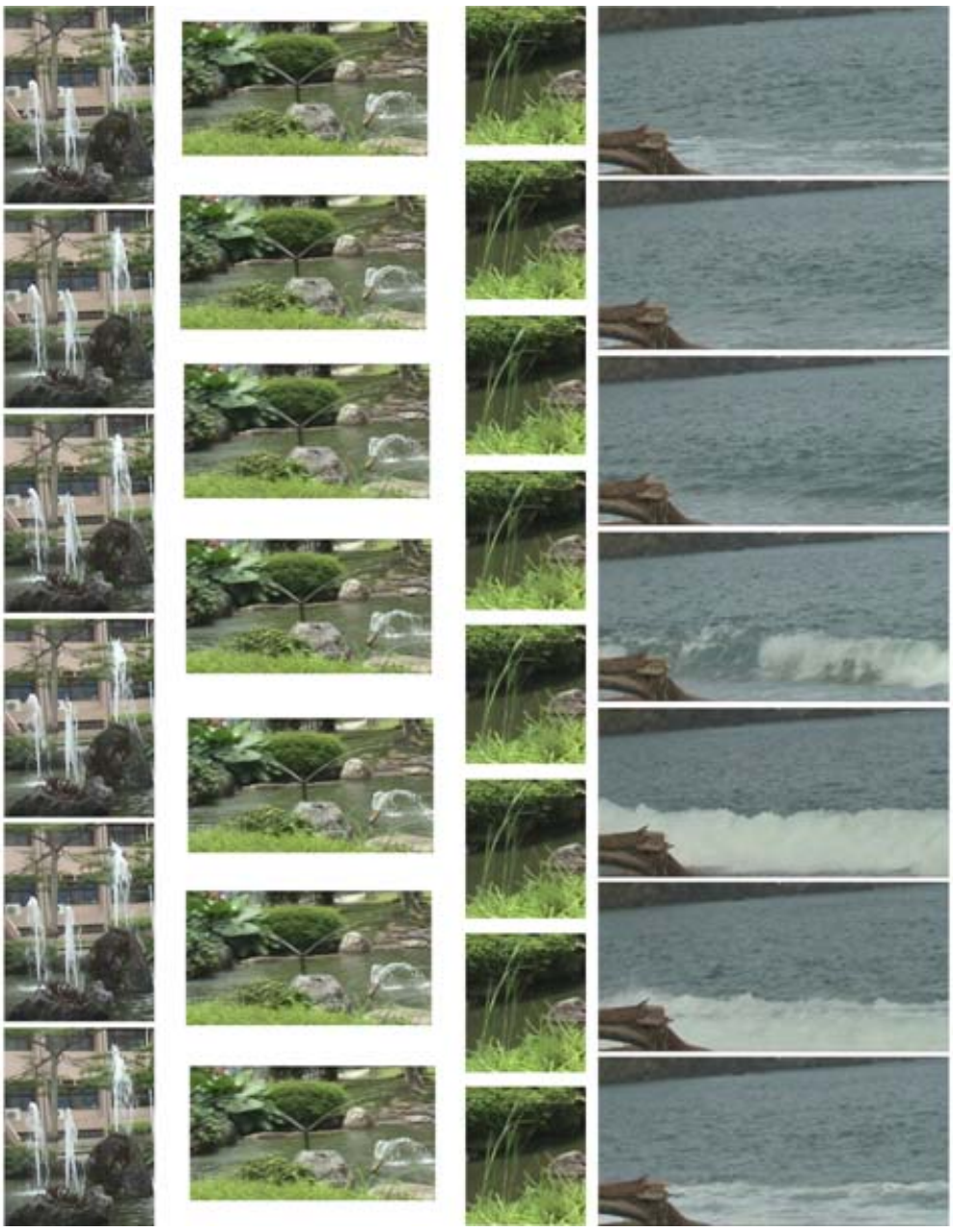

Figure 11: Four video texture examples. Every 4th, 4th, 3rd, and 35th frame of fountain1, fountain2, straws, and wave video textures are shown from left to right, respectively.

video synthesis using graph cuts. In *Proc. SIGGRAPH 03*, San Diego, California, USA, July, 2003, pages 277-286.

[9] S. Nayar. Catadioptic omnidirectional cameras. In *Proc. CVPR'97*, San Jaun, Puerto Rico, USA, June, 1997, pages 482-488.

[10] U. Neumann, T. Pintaric and A. Rizzo. Immersive panoramic video. In *Proc. ACM MULTIMEDIA'00*, pages 493-494, 2000.

[11] A. Schödl, R. Szeliski, D. H.Salesin and I. Essa. Video textures. In *Proc. SIGGRAPH 2000*, New Orleans, Louisiana, USA, July, 2000, pages 489-498.

[12] Y. Wexler, E. Shechtman and M. Irani. Space-time completion of video. In *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, pages 463-476, 2007.