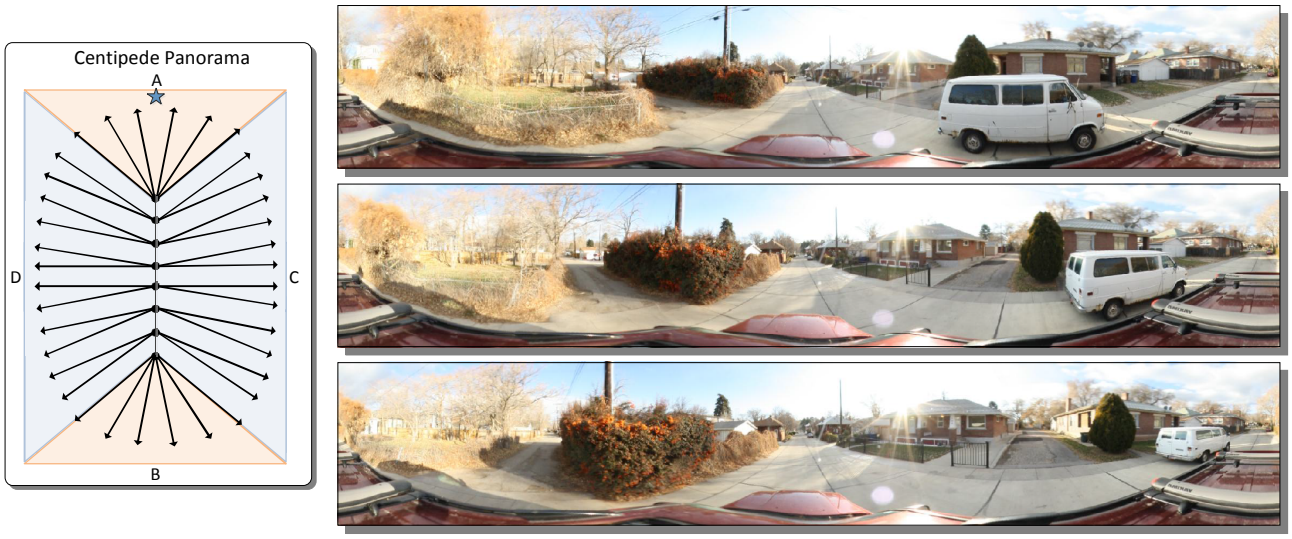
A Flexible Framework for Fusing Image Collections into Panoramas

Wathsala Widanagamaachchi, Paul Rosen, Valerio Pascucci
Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, UT, USA
Email: wathsy@sci.utah.edu, prosen@sci.utah.edu, pascucci@sci.utah.edu



(a) Centipede constructor (b) Series of centipede panoramas

Fig. 1. The centipede panorama is a hybrid of the 360° and street panoramas with wide angle views towards the end (sections A and B) and composite viewpoints in-between (sections C and D). The blue star indicates the middle of the panoramic image.

***Abstract*—Panoramas create summary views of multiple images, which make them a valuable means of analyzing huge quantities of image and video data. This paper introduces the *Ray Graph* - a general framework for panorama construction. With rays as its vertices, the Ray Graph uses its edges to specify a set of coherency relationships among all input rays. Consequently, by using a set of simple graph traversal rules, a diverse set of panorama structures can be enumerated, which can be used to efficiently and robustly generate panoramic images from image collections. To demonstrate this framework, we first use it to recreate both 360° and street panoramas. We further introduce two new panorama models, the centipede panorama - a hybrid of 360° and street panoramas, and the storytelling panorama - a time encoding panorama. Finally, we demonstrate the flexibility of this framework by enabling interactive brushing of panoramic regions for removal of undesired features such as occlusions and moving objects.**

***Keywords*-panoramas; image fusion; image analysis;**

I. Introduction

A surge of image and video data has surfaced as cameras have become increasingly affordable. Managing, displaying and analyzing this abundance of data continues to be challenging. One promising approach for addressing these challenges is panoramic imaging. Panoramas are wide-area or wide-angle views of a physical space which enable a photo-realistic environment that is more comprehensive than a conventional image. Panoramic imaging has its first roots in 19th century paintings, but more recently, they have been used in a variety of applications such as virtual reality [1], telepresence [2] and situational awareness [3].

Most panorama techniques achieve continuity in the output image by maintaining smooth transition of view position and direction between neighboring pixels. For example, the 360° panorama [4] consists of single viewpoint whose continuity comes from the smooth change in viewing direction. The street panorama [5], on the other hand, fuses multiple images by collecting parallel rays from nearby viewpoints. A wide variety of research continues to be performed in the area of panorama generation; however, there is still a need for a general technique which can assemble a diverse variety of panoramas.

In this paper, we present a new approach to generalize the construction of panoramas. The core of our methodology is the Ray Graph structure, which defines coherency relationships between two rays from a single or multiple images. These coherency relationships are then extended to arbitrarily large sets of images where the relationships between all sets of rays are considered to be a weighted graph. Traversals within these weighted graphs are then defined to create *arbitrarily-shaped panoramas* from input image data. Our approach enables quick and easy creation of both 360° and street panoramas.

In addition, we develop two new forms of panorama, the centipede and storytelling panoramas. The centipede panorama is a hybrid of 360° and street panoramas, which stretches the viewpoint of the 360° panorama into a view segment providing a longitudinal and circumscribing view of the physical space. Fig. 1(a) shows an example centipede panorama built from images taken from a car driving down the street. The centipede panorama captures the objects on the sides of the road using nearly parallel views and captures a wide field-of-view, both forward and backward, resulting in an omni-directional output image. The storytelling panorama captures temporal events into a panorama for summarizing the past.

Finally, to further demonstrate the potential of our framework, we use this Ray Graph structure for interactive modification of panoramas to allow removal of undesirable features.

In summary, the contributions of this paper are:

- Ray Graph model, which unifies panorama construction into a single flexible framework for creating panoramas of arbitrary shape;
- Two new forms of panoramas, the centipede panorama and the storytelling panorama, which demonstrate the utility of this framework; and
- A semi-automatic panoramic modification method, which uses brushing of panoramic regions to remove undesirable features in panoramas.

II. Related Work

A. Panoramic Images

360° panoramas [4] encode all directions visible from a single viewpoint. The single viewpoint constraint was later relaxed by mosaicing nearly coincident viewpoints [6]. Subsequent improvements to mosaicing include better correspondence finding [7], reduced ghosting from parallax [8], and using depth for better stitching [9], [10]. 360° panoramas produce good summary views but lack flexibility in construction. However, they are reproducible by Ray Graphs.

Street panoramas [5], [11], route panoramas [12], and multiple center-of-projection images [13] integrate multiple viewpoints using a vertical push-broom camera to capture scenes, such as building facades along a street. Later, by applying a crossed-silt projection [14] along with scene geometry estimations, the perspective of the street panorama was dynamically varied to compensate for distortions [15], [16]. Just as with 360° panoramas, street panoramas are a natural subset of panoramas reproducible by Ray Graphs.

The most closely related work for time-varying panoramas is that of resampling a video cube, a stack of images gathered by a video camera, either stationary or moving, along a continuous path [17], [18]. Arbitrary cuts through the video cube are used to produce time-varying panoramas. This video cube has been used for impressionism, cubism and abstract aesthetic video effects [19]. Space-time scene manifolds [20] similarly allow cuts through what amounts to a video cube. There is also a body of work for generating 360° video panoramas of scenes with periodic motion [21]. In addition to these, time-lapse approaches have also been used to produce time-varying panoramas for fixed viewpoints [22]. These systems, while not as general as Ray Graphs, still observe coherency between rays when producing imagery.

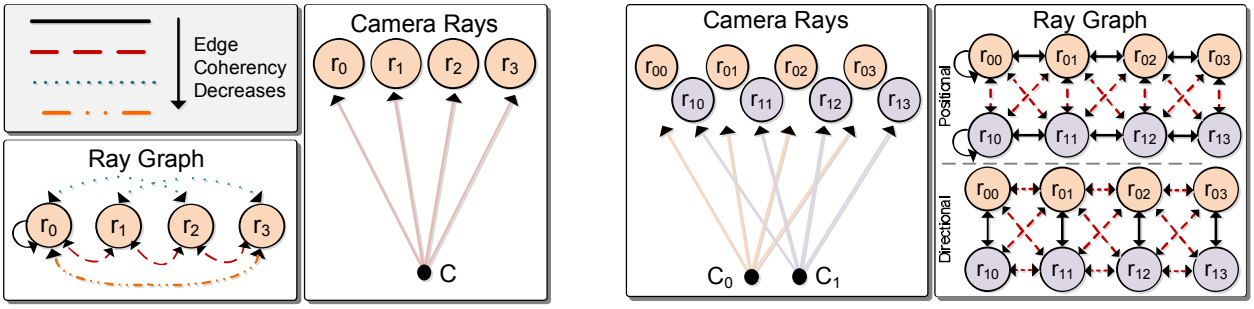
There are also a number of methods available to capture single shot panoramic images. The most popular means is by using a camera with special optics such as hexagonal pyramidal mirrors [23], paraboloidal mirrors [24], hyperbolic mirrors [2], using a camera with wide field-of-view lens such as a fisheye lens [25], or multiple synchronized cameras [26]. The images produced can not only be used in their raw form, but also as input data to create other types of panoramas. For example, we use a hyperbolic mirror to capture our data (see Section V-A).

B. Ray-based Image Representations

Some of the first ray-based representations of data from multiple images were the light field and lumigraph [27], [28]. These methods capture multiple images into a dense set of rays which are later used to reconstruct novel views of the data. Though these novel views were generally not panoramas, there is no fundamental limitation in these approaches which would prevent such reconstructions. Concentric mosaics is another ray-based representation which can render novel views by combining rays at rendering time [29]. We consider these representations to be a subset of the Ray Graph, since the Ray Graph could also be used to encode light field data and reconstruct novel conventional views. Additionally, since it can use both dense and sparse sets of rays to produce panoramas, we consider the Ray Graph a more flexible representation.

General Linear Cameras (GLCs) [30], [31], [32] partition an image plane into triangular images. Each GLC is constructed from three given rays, so it offers some ray modeling flexibility. By blending rays of neighboring GLCs [33], a continuous ray space is generated for images with the perspective varying smoothly. This gives GLCs the ability to model potentially complex camera structure. Furthermore, GLCs allow for specifying cameras at a coarser granularity than the Ray Graph. However, the Ray Graph gives a greater flexibility in design than GLCs.

The Graph Camera [34] combines sparse sets of viewpoints to produce panoramic images by generalizing camera rays into piecewise linear splines and connecting them. To the best of our knowledge, this is the only panoramic imaging technique *not* reproducible using a Ray Graph.



(a) Partial Ray Graph for directional edges only

(b) Partial Ray Graph for positional/directional (top/bottom) edges

Fig. 2. Partial Ray Graphs with positional and directional coherencies. A complete Ray Graph would connect all nodes.

III. Ray Graphs

A Ray Graph is an implicit weighted graph structure which defines the coherency between all pairs of rays within a set of images. In the graph, rays serve as the vertices which are connected by edges weighted by tuples whose members each refer to a coherency relationship: directional, spatial, temporal, etc. To produce output images from these weighted graphs, simple traversal algorithms are created that walk the Ray Graph selecting rays to be used in the output image. The following explanations are within the context of a 1D output image (i.e. a row of pixels). However, the actual implementation is for a 2D image.

A. Coherency Relationships

Directional Coherency: The directional coherency between rays within a single image can be considered strong if both rays point in the same direction. We define the strength of the directional coherency (s_D) between the two rays r_0, r_1 using the following: $s_D(r_0, r_1) = 1 - \frac{r_0 \cdot r_1}{\|r_0\| \|r_1\|}$. This way, two neighboring rays have a stronger directional coherency (closer to 0) than the rays pointing in opposite directions (towards 2). Fig. 2(a) shows a single camera Ray Graph. A ray, such as r_0, has the strongest directional coherency with itself. When r_0 is considered with r_1, they have a slightly weaker directional coherency, while rays further apart, like rays r_0 and r_3, have very low directional coherency.

Spatial Coherency: Comparing the direction of rays is sufficient for describing the relationship between rays of a single image, but it is insufficient to describe the relationships between multiple images. For multiple images, the relative positions of their centers of projection have to be considered as well. To support this, the Euclidean distance between two viewpoints is used as the second coherency measure. Thus the spatial coherency (s_P) between two viewpoints C_0, C_1 is found by using: $s_P(C_0, C_1) = \|C_0 - C_1\|$. Given this metric, images taken next to each other have a stronger spatial coherency (closer to 0) than the images farther away. Fig. 2(b) shows a Ray Graph involving two images where the self-referencing spatial coherency is 0. Still, the spatial coherency between the two cameras is strong since they are relatively close to one another.

Temporal Coherency: Ray Graphs can encode temporal events as well. For example, a pedestrian walking with a handheld camera pauses at a stop light. Such an event would be lost by the two aforementioned metrics. To capture temporal coherency (s_T) between rays from two times t_0, t_1 we define: $s_T(t_0, t_1) = t_1 - t_0$. This bi-directional metric allows consecutive times to have a stronger temporal coherency (closer to 0) than the rays from distant times while also encoding future (positive valued) and past (negative valued) events.

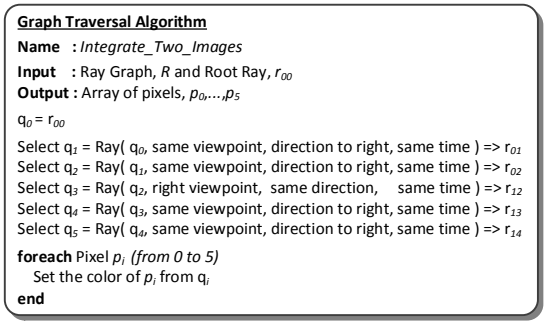
The final edge coherency is the tuple formed by $\{s_D, s_P, s_T\}$.

B. Retrieving Images from a Ray Graph

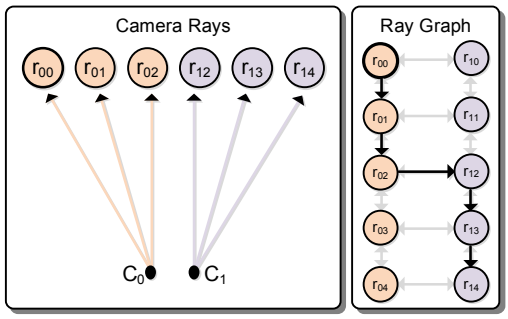
An output image is formed by traversing a path through the Ray Graph. Given that the graph is dense, there are many potential paths. Therefore, producing images requires a meaningful traversal algorithm to be specified. Our traversal algorithms are defined with the use of a simple set of rules. These rules are specified in terms of a root ray and a reference coordinate system (an origin, view direction/root ray, and up direction) that is associated with the root pixel in the output image. Starting from the root ray, the graph is traversed, selecting the rays associated with each pixel in the output image. At each pixel, the algorithm queries the Ray Graph for the closest matching ray for a desired edge condition (specified by ray direction, viewpoint and time), and the selected ray is used for that pixel.

Given a reference coordinate system, the user is given a set of simple commands for selecting the next ray location. For position changes, the options include left/right, forward/backward, and up/down with a user specified distance. For direction changes, the options include left/right and up/down with a user specified angle. For time, forward and backward options are given for a user specified amount of time. The user is also given access to the reference coordinate system for the option of more complex paths, though none of our panoramas required that functionality.

Fig. 3 shows a simple graph traversal where two images are integrated seamlessly by crossing the edge between r_{02} and r_{12}. The algorithm for this traversal is outlined in *Integrate_Two_Images* (Fig. 3(a)). The traversal starts from the root ray, q_0/r_{00} and obtains rays for each pixel in the



(a) Pseudo-coded graph traversal algorithm



(b) Output image rays (left) and graph traversal (right)

Fig. 3. The two images from Fig. 2(b) are integrated seamlessly, beginning at the root ray (r_{00}), first by varying view direction, followed by viewpoint and again by view direction.

output image. It starts by finding the ray which has the same viewpoint, time and a right direction to q_0 (i.e. *Ray(q_0, same viewpoint, direction to right, same time)*. This newly identified ray is used to obtain the next ray, and so on. Finally, colors for each pixel are set by using the rays selected.

C. Implementation

Ray Graph is stored implicitly with edge weights calculated at runtime based on queries set forth by the traversal algorithm. The input to the Ray Graph is a set of images and their associated camera parameters, including intrinsic and extrinsic parameters. When traversing the graph, a desired ray is specified by its viewpoint, direction and time. That ray is then used to query the Ray Graph for the closest matching ray and associated pixel color, from the ray database. When queried, the Ray Graph first identifies images with similar time to the query, next within that time the closest viewpoint is selected, and then the best ray within that viewpoint is identified. If desired data is unavailable, the Ray Graph query will look for the next best data available. This is done by expanding the range of acceptable times, then viewpoints, and finally ray directions.

To perform these queries, well known data structures and algorithms are used. *Temporal coherency* (s_T) relationships are processed by storing the viewpoints in a list ordered by time and accessing them appropriately. *Spatial coherency* (s_P) relationships are found by storing all viewpoints within a k-nearest neighbor graph. When queried, the graph returns an ordered list of viewpoints that best matches the desired location. Finally, *directional coherency* (s_D) relationships between two rays are found by using the intrinsic and extrinsic camera parameters. The direction of the desired ray is simply projected and the matching ray is returned with subpixel interpolation.

IV. Panorama Construction & Modification

We demonstrate the use of the Ray Graph framework by creating four types of panoramas: two pre-existing panoramas and two new panoramas. In addition to creating arbitrarily-shaped panoramas, Ray Graph structure can also be used to modify them. As such, we present an interactive panoramic modification method for removing undesirable features.

A. 360° Panorama

The graph traversal algorithm used for creating the 360° panorama maintains directional coherency by varying the view direction (see Fig. 4(a)). First, we select the root ray to be the center pixel of the output image. Then the algorithm traverses edges by varying the view direction between neighboring rays within the same viewpoint. Fig. 4(b) shows a series of examples.

B. Street Panorama

The traversal algorithm used for constructing the street panorama maintains directional and spatial coherency by selecting rays in the same direction, from different viewpoints (see Fig. 5(a)). The algorithm starts at a selected root ray (left most pixel in the output image) and traverses along parallel rays until it reaches the end of the image. Example images are shown in Fig. 5(b).

C. Centipede Panorama

The centipede panorama, whose name is derived from the visualization of its rays (Fig. 1(a)), integrates both wide-angle and wide-area views. As seen in Fig. 1(a), the traversal algorithm used for the centipede panorama is divided into four sections. The section A is set to be the middle of the output image. The number of output image pixels associated with each section is specified at construction time.

Sections A and B, each uses a single viewpoint where view direction is varied between neighboring rays. The angle between the rays used depends on the number of pixels available to each section, along with their desired field-of-views. Edges in sections C and D alternate between view directional variation and spatial variation. There, the number of intermediate viewpoints used and the rays per viewpoint are adjustable and are partially based on the number of pixels and field-of-view available. To avoid redundancy within the output image, the combined field-of-view for all sections is set to be 360°. Fig. 6 shows an example where the number of intermediate viewpoints is modified and the field-of-view for the viewpoints is adapted.

Fig. 1(b) shows a series of centipede panorama images that capture a forward and backward view, along with an

(a) 360° constructor (b) Series of 360° panoramas

Fig. 4. A series of 360° panoramas constructed by varying the view direction within a single viewpoint.

(a) Street constructor (b) Series of street panoramas

Fig. 5. A series of street panoramas constructed with parallel rays traversed by varying the viewpoint.

extended view of the sides of the road, resulting in an omni-directional panoramic image. Here, each section (A, B, C, and D) receive one quarter of the output image pixels. Since centipede panorama maintains a 360° field-of-view while integrating multiple views, it combines the advantages of both 360° and street panoramas.

Fig. 6. Centipede panoramas where the number of intermediate viewpoints is increased (top to bottom) and the field-of-view for the viewpoints is adapted.

D. Storytelling Panorama

The final form of panorama we introduce, the storytelling panorama, is used to fuse a set of images that vary over time. This panorama has two sections, one for the present and the other for the past (see Fig. 7(a)).

The middle of the output image captures the present time (section A) by using rays with view directional variation. The remainder of the image, corresponding to the past (section B), is where composite viewpoints with small directional variations are separated by increasingly ordered temporal variation edges, moving further into the past. As the panorama moves deeper into time, the proportion of rays from the distant past is decreased. The result is that older history occupies an ever shrinking portion of the output image, making space for newer information. Fig. 7(b) shows a series of storytelling panorama images as time advances (top to bottom).

Unlike the above example, if there is a very distinctive landscape, that landscape information will be retained throughout the storytelling panorama. However, it will occupy increasingly smaller portions of the image. Moving objects will occupy a smaller or a larger portion in section B depending

(a) Storytelling constructor (b) Series of storytelling panoramas

Fig. 7. As the storytelling panorama moves through time (top to bottom), time is mapped from the center of the image (current time) to the outer edge (oldest time) with smaller view directional variations for past events.

upon the amount of time they are visible in the original input images.

E. Panoramic Modification

In a given panoramic image, there can be any number of undesirable features visible due to capture conditions (i.e. lens flares, occlusions, or moving objects). The Ray Graph can easily be extended to support interactive panoramic modification. Our method consists of a stroke-based interface that allows repeated brushing of panoramic regions to remove such features. As a user selects a particular region on the panorama, the rays represented by those areas are flagged and their weights are increased, causing the Ray Graph to attempt to avoid them in the future. After each brush, the output image is re-calculated, with flagged rays avoided and replaced with unflagged rays. The user can repeat the brushing as many times as needed, flagging additional rays to be excluded from the output image. Fig. 8 shows several such examples where this semi-automatic modification method has been used to remove occluding objects from panoramas.

V. Results & Discussion

A. Data

Three datasets were used in our examples. The first 2 datasets were collected using the *0-360 Panoramic Optic*, a hyperbolic panoramic lens system, attached to a Canon Rebel XS DSLR camera. The first dataset (Fig. 4(top), Fig. 5(top), Fig. 6, Fig. 7, Fig. 8, Fig. 9 and supplemental material) consisted of 1,300 panoramic images (25 GB and ∼5.45 billion rays) captured by attaching the system to a tripod and dolly. The second dataset (Fig. 1, Fig. 4 bottom, Fig. 5(middle), Fig. 5(bottom) and supplemental material) contained 127 images (1.7 GB of data) captured using the same imaging system mounted on a vehicle. The final dataset (Fig. 4(middle) and supplemental material) is the publicly available New College [35] dataset, captured using a ladybug camera attached to a robot. The dataset contains over 39,000 images (9.7 GB of data), but we only experimented with a subset of 300 images. As the first two datasets were taken in more active environments, they contained more moving objects like vehicles and pedestrians than the third. Apart from those the first dataset also contained some lens flares. For creating a denser dataset, optical flow [36] was used (see Section V-C). Although only a single camera was used to capture each of these datasets, there is no limitation in the Ray Graph to handle images acquired from multiple cameras with different resolutions.

B. Performance

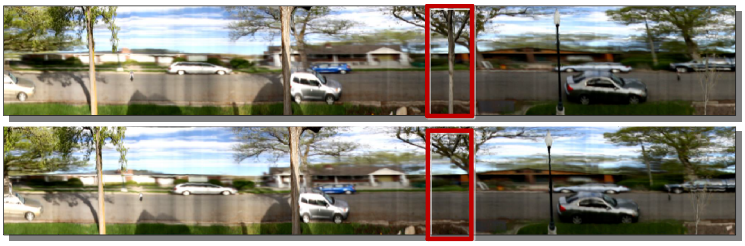
Our system operated on a PC with a 3.2GHz Intel i7 CPU and 4.0GB of RAM.

Data Preprocessing: Before being fed into the Ray Graph system, data must be preprocessed by the optical flow and camera pose systems. The optical flow typically takes 90 seconds to calculate and a few seconds to generate a new image. The pose estimation takes 15-60 seconds per image, depending upon a number of conditions. It is certainly important to speed up this processing time, but it remains outside the immediate scope of this work.

Ray Graph: The creation of the Ray Graph structure requires a minimal amount of load time, 5-30 seconds depending upon the number of images in the database. The speed of the construction algorithms depends heavily upon the complexity of analysis needed during this construction, but all of our methods are fast, producing the 2D grid of rays between a few frames per second up to hundreds of frames per second.

Rendering Images: The final component of the pipeline is the rendering of the output image. Once the Ray Graph is specified, a common functionality is used to render the final

(a) First example of occlusion removal before (top) and after (bottom)



(b) Second example of occlusion removal before (top) and after (bottom)

(c) Zoom of Fig. 8(a)

Fig. 8. Two examples where brushing has been used to remove occlusions. The regions of modification are shown in the red boxes.

image. The color value for each ray in the output image is retrieved from the original source data. We built a small caching system for accessing these image data. Cold cache performance was in the order of minutes, but warm cache performance was less than 1 frame per second or faster for all 4 types of panorama at 1280×720 output resolution. Still, cache misses were extremely expensive in our system. With additional design considerations to the database, this process could be improved to tens or hundreds of frames per second.

C. Limitations

Ray Graphs are capable of produce images at a high-quality. However, some of the panorama constructors are more or less sensitive to the quality of the input data. For example, the street panorama is sensitive to the density of viewpoint spacing. On the other hand, the centipede panorama needs fewer input images (when compared to street panorama), making it more robust to sparse input data.

This lead to temporal and spatial sparsity of data being a potential limitation. Our photographs were taken at both temporally and spatially sparse locations. To compensate for this, we used optical flow to generate intermediate viewpoints. Fig. 9 shows a comparison of panoramas with and without the optical flow. Both images are similar from an information content perspective, but the visual quality is significantly better with optical flow. This limitation and the need for optical flow is primarily due to our capturing system and would be alleviated with better sensors.

The other limitation to the Ray Graph system is its sensitivity to the quality of pose information. The output images tend to be robust to low frequency errors in the camera pose, but sensitive to high frequency errors. In other words, two viewpoints next to each other must have relatively well coordinated poses; however, if there is some global distortion, image quality will still remain high.

VI. Conclusion

In conclusion, we have presented the Ray Graph, a system for unifying panorama construction. One of our intended future applications for the Ray Graphs framework is real-time image fusion from multiple sensors. In such an environment, the Ray Graph's ability to handle both sparse and dense image sets in producing arbitrarily-shaped panoramas is extremely advantageous.

We see many avenues of future work which will expand the capabilities of Ray Graphs. One obvious missing piece of the current Ray Graph system is the lack of depth in the representation. This approach is already common among street panoramas, where the depth is used to automatically adjust the perspective. As such, we would like to enhance the Ray Graph with additional coherency relationships including depth measurements. We would also like to pursue the idea of developing automatic panorama construction techniques. These methods would use simple rules of operation to perform some kind of search or an optimization on the Ray Graph. However, this also needs further study to determine the best possible traversal algorithms for encoding the most information into meaningful panoramas, along with the appropriate user studies to test comprehension and usability.

Acknowledgment

This work is supported in part by NSF OCI-0906379, NSF OCI-0904631, DOE/NEUP 120341, DOE/MAPD DESC000192, DOE/LLNL B597476, DOE/Codesign P01180734, and DOE/SciDAC DESC0007446.

References

[1] D. Chapman and A. Deacon, "Panoramic imaging and virtual reality – filling the gaps between the lines," *ISPRS J. of Photogram. and R. Sensing*, vol. 53, no. 6, pp. 311–319, 1998.

(a) First example of centipede panorama with (top) and without (bottom) optical flow

(b) Second example of street panorama with (top) and without (bottom) optical flow

(c) Zoom of Fig. 9(b)

Fig. 9. A comparison of results with and without using optical flow to create denser data. Although, optical flow does not change the information content of images, it does result in higher quality images.

[2] Y. Onoe, K. Yamazawa, H. Takemura, and N. Yokoya, "Telepresence by real-time view-dependent image generation from omnidirectional video streams," *Computer Vision and Image Understanding*, vol. 71, no. 2, pp. 154 – 165, 1998.

[3] M. Plumlee, C. Ware, R. Arsenault, and R. T. Brennan, "Panoramic images for situational awareness in a 3d chart-of-the-future display," in *U.S. Hydro. Conf.*, 2005.

[4] S. E. Chen, "Quicktime VR: An image-based approach to virtual environment navigation," in *SIGGRAPH '95*, 1995, pp. 29–38.

[5] A. Roman, G. Garg, and M. Levoy, "Interactive design of multi-perspective images for visualizing urban landscapes," in *VIS '04*, 2004, pp. 537–544.

[6] R. Szeliski and H.-Y. Shum, "Creating full view panoramic image mosaics and environment maps," in *SIGGRAPH '97*, 1997, pp. 251–258.

[7] M. Brown and D. G. Lowe, "Recognising panoramas," in *ICCV '03*, 2003.

[8] S. B. Kang, R. Szeliski, and M. Uyttendaele, "Seamless stitching using multi-perspective plane sweep," 2004.

[9] J. Gao, S. J. Kim, and M. Brown, "Constructing image panoramas using dual-homography warping," in *IEEE Conference on CVPR '11*, 2011, pp. 49–56.

[10] K. C. Zheng, S. B. Kang, M. F. Cohen, and R. Szeliski, "Layered depth panoramas," in *CVPR*. IEEE Computer Society, 2007.

[11] A. Agarwala, M. Agrawala, M. Cohen, D. Salesin, and R. Szeliski, "Photographing long scenes with multi-viewpoint panoramas," *ACM TOG*, vol. 25, no. 3, pp. 853–861, 2006.

[12] J. Zheng and M. Shi, "Scanning depth of route panorama based on stationary blur," *International Journal of Computer Vision*, vol. 78, pp. 169–186, 2008.

[13] P. Rademacher and G. Bishop, "Multiple-center-of-projection images," in *SIGGRAPH '98*, 1998, pp. 199–206.

[14] A. Zomet, D. Feldman, S. Peleg, and D. Weinshall, "Mosaicing new views: The crossed-slits projection," *IEEE Trans. on Pattern Anal. and Machine Intel.*, vol. 25, no. 6, pp. 741–754, 2003.

[15] A. Roman and H. P. A. Lensch, "Automatic multiperspective images," in *EGSR '06*, 2006, pp. 161–171.

[16] A. Rav-Acha, G. Engel, and S. Peleg, "Minimal aspect distortion (mad) mosaicing of long scenes," *Int. J. Comput. Vision*, vol. 78, no. 2-3, pp. 187–206, Jul. 2008.

[17] S. M. Seitz and J. Kim, "Multiperspective imaging," *IEEE CG&A*, vol. 23, no. 6, pp. 16–19, 2003.

[18] E. P. Bennett and L. McMillan, "Proscenium: a framework for spatio-temporal video editing," in *ACM Multimedia*, 2003, pp. 177–184.

[19] A. W. Klein, P.-P. J. Sloan, A. Finkelstein, and M. F. Cohen, "Stylized video cubes," in *SCA '02*, 2002, pp. 15–22.

[20] Y. Wexler and D. Simakov, "Space-time scene manifolds," in *ICCV '05*, 2005, pp. 858–863.

[21] A. Agarwala, K. C. Zheng, C. Pal, M. Agrawala, M. Cohen, B. Curless, D. Salesin, and R. Szeliski, "Panoramic video textures," *ACM TOG*, vol. 24, pp. 821–827, 2005.

[22] M. Terry, G. Brostow, G. Ou, J. Tyman, and D. Gromala, "Making space for time in time-lapse photography," in *SIGGRAPH Technical Sketches*, 2004.

[23] T. Kawanishi, K. Yamazawa, H. Iwasa, H. Takemura, and N. Yokoya, "Generation of high-resolution stereo panoramic images by omnidirectional imaging sensor using hexagonal pyramidal mirrors," in *Intern. Conf. on Patt. Recog.*, 1998, pp. 485–489.

[24] S. K. Nayar, "Catadioptric omnidirectional camera," in *CVPR '97*, 1997, pp. 482–488.

[25] Y. Xiong and K. Turkowski, "Creating image-based VR using a self-calibrating fisheye lens," in *CVPR '97*, 1997, p. 237.

[26] M. Uyttendaele, A. Criminisi, S. Kang, S. Winder, R. Szeliski, and R. Hartley, "Image-based interactive exploration of real-world environments," *IEEE CG&A*, vol. 24, no. 3, pp. 52 – 63, 2004.

[27] M. Levoy and P. Hanrahan, "Light field rendering," in *SIGGRAPH '96*, 1996, pp. 31–42.

[28] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen, "The lumigraph," in *SIGGRAPH '96*, 1996, pp. 43–54.

[29] H.-Y. Shum and L.-W. He, "Rendering with concentric mosaics," ser. SIGGRAPH '99, 1999, pp. 299–306.

[30] J. Yu and L. McMillan, "General linear cameras," in *ECCV*, vol. 2, 2004, pp. 14–27.

[31] J. Ponce, "What is a camera?" in *CVPR '09*, 2009, pp. 1526 –1533.

[32] J. Yu and S. Leonard, "Muliperspective modeling, rendering, and imaging," in *Proceedings of Eurographics 2010, Computer Graphics Forum*, 2010, pp. 227–246.

[33] J. Yu and L. McMillan, "Modeling reflections via multiperspective imaging," in *CVPR '05*, vol. 1, 2005, pp. 117–124.

[34] V. Popescu, P. Rosen, and N. Adamo-Villani, "The graph camera," *ACM TOG*, vol. 28, pp. 158:1–158:8, December 2009.

[35] M. Smith, I. Baldwin, W. Churchill, R. Paul, and P. Newman, "The new college vision and laser data set," *International J. of Robotics Research*, vol. 28, no. 5, pp. 595–599, May 2009.

[36] C. Liu, "Exploring new representations and applications for motion analysis beyond pixels," Ph.D. dissertation, Massachusetts Institute of Technology, 2009.

Supplementary materials for the paper titled "A Flexible Framework for Fusing Image Collections into Panoramas"

Wathsala Widanagamaachchi, Paul Rosen, Valerio Pascucci
Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, UT, USA
Email: wathsy@sci.utah.edu, prosen@sci.utah.edu, pascucci@sci.utah.edu

This document serves as a supplementary document to further expand upon the capabilities of the Ray Graph structure.

I. 360° and Street Panoramas

We first provide some additional examples for 360°and street panoramas. Fig. S.1 shows some results of 360°panoramas from two different datasets. These have been created from a Ray Graph traversal which maintains the directional coherency by varying the view direction. Fig. S.2 shows a longer result of a street panorama constructed using a Ray Graph traversal which collects parallel rays.

II. Centipede Panorama

The centipede panorama is a hybrid of 360°and street panoramas. As described in Section IV-C of the paper, the Ray Graph traversal used for generating this panorama is divided into four sections. There, sections C and D alternate between view directional variation and spatial variation, where the number of intermediate viewpoints used and the rays per viewpoint can be adjusted. Fig. S.3 shows a series of centipede panoramas where the number of viewpoints is decreased (top to bottom) and the number of rays per intermediate viewpoint is increased to adapt (top to bottom).

III. Storytelling Panorama

Fig. S.4 and Fig. S.5 shows two examples of storytelling panoramas created from two different datasets. As the panorama moves through time (top to bottom), the history of what has been seen is recorded in the periphery of the panorama.

IV. Interactive Editing

Our final example, Fig. S.6 shows before and after results of a street panorama where brushing has been used to remove occlusions within the panorama. Here, repetitive brushing of the same region has been used to successfully remove occluded objects from the original panorama.

Fig. S.1 : Examples of 360^{o} panoramas constructed using a Ray Graph traversal which selects the rays by varying the view direction.

Fig. S.2 : A longer street panorama example constructed using a Ray Graph traversal which moves along parallel rays between neighboring viewpoints.

Fig. S.3 : Series of centipede panoramas where the number of intermediate viewpoints is decreased (top to bottom) and the number of rays per viewpoints is adapted.

Fig. S.4 : Series of storytelling panoramas constructed using a Ray Graph traversal. As the panorama moves through time (top to bottom), the history of what has been seen is recorded in the periphery of the panorama.

Fig. S.5 : A series of storytelling panoramas constructed using a Ray Graph traversal.

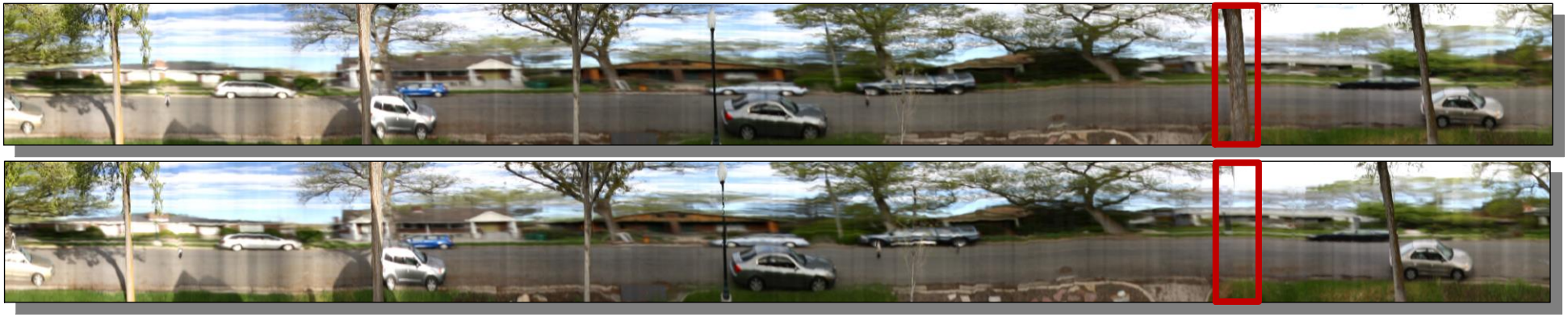


Fig. S.6 : Before and after (top and bottom) example of street panorama, where brushing has been used to remove occlusions.