A Parallel Algorithm for Mosaicing Near Ground UAV Videos

Mohamed A. Helala
University of Ontario Institute of Technology

April 7, 2013

Abstract

Image mosaicing is the process of stitching a set of overlapped images to produce a panoramic image. This enables several applications to overcome the limited field of view of cameras. Image mosaicing is a fundamental process in aerial imagery and is a simple task for far ground imagery where the scene has less depth variations and is considered planar. However, near ground aerial imagery will have clear parallax effects resulted from the occlusion of man-made structures with respect to the camera motion. This project addresses the mosaicing of near ground aerial videos and handle the parallax problem by extending the algorithm in [1] to recover a view dependent mosaic. The extension consists of two stages, a data processing stage and a mosaic generation stage. The input to the data processing stage consists of a sequence of video frames with a camera motion trajectory containing the camera parameters of each frame. This trajectory is clustered into a set of overlapped clusters and an interpolated mosaic is constructed for each cluster using Plane Sweep Stereo (PSS) [2, 1, 3]. The mosaic generation stage then, receives the generated set of mosaics and the pose of a final virtual camera looking at the scene from a certain viewpoint and constructs a final view specific mosaic by projecting and stitching the interpolated mosaics onto the virtual camera.

Plane sweep stereo recovers a depth map from a set of input source cameras (A camera mean pose parameters and image plane) and a mosaic image is defined according to the recovered map. PSS starts by interpolating the poses of the input source cameras to define a reference camera. Then, It discretizes the scene 3D space by defining a set of sweep planes in a certain depth range and parallel to the image plane of the reference camera. The scene depth map is recovered by applying focus analysis to select the focus depth of each pixel on the reference camera image plane.

PSS is used in the data processing stage of our project and can be very slow in case of scenes with large depth range. Also, PSS will be applied on each cluster of a possibly long camera trajectory. This highly increases the computational cost of the data processing stage compared to the mosaic generation stage. So, the focus of this project will be on parallelizing the data processing stage of the discussed algorithm.

(a) Generating a good panorama from parallax free (planar) images.

(b) Undesirable artifacts (object repetition and ghost effects) due to camera motion parallax.

Figure 1: Comparison of two panoramas constructed from images without (a) and with (b) motion parallax. The amount of occlusion caused by parallax is indicated by the numbered objects. (Source [3])

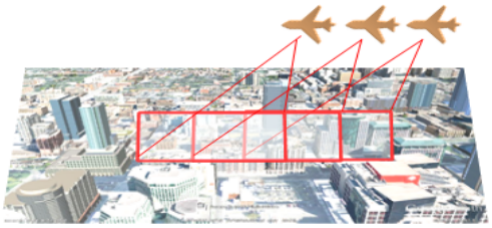
1 Introduction

Nowadays, Unmanned Aerial Vehicles (UAVs) have been increasingly used in aerial imagery. They were used for collecting aerial images and videos from several altitudes above ground. The collected data supports several tasks in applications such as fire monitoring, search and rescue, and forest surveillance [4, 5]. Such applications benefit from the ability to extend the Field of View (FOV) of the captured video data. This drives the interest in mosaicing techniques to stitch together a sequence of images and construct large image maps.

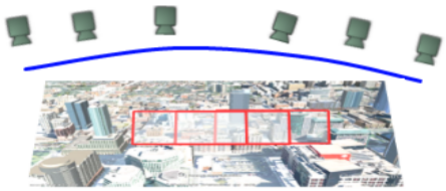
The mosaicing techniques used for aerial videos differ in terms of the altitude of the captured scene. Videos taken from far ground locations will have very small depth variations between successive frames so, the scene is assumed planar and traditional mosaicing techniques [6, 7] such as panorama stitching can work very well. In the other hand, near ground video sequences will have very noticeable depth variations between successive frames due to motion parallax which can be defined [3] as the relative displacement of scene objects with respect to camera motion. This causes occlusion of objects between successive frames (Figure 1b) and require the development of special mosaicing techniques that take care of the resulted occlusion for constructing large mosaics.

This work addresses the parallel implementation of a mosaicing technique that extends the work in [1] to construct view dependent mosaics from near

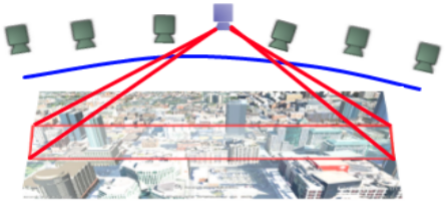
ground aerial videos. A view dependent mosaic is a mosaic created from a virtual camera location looking at the scene from a specific viewpoint. By changing the location of the virtual camera, one can obtain a different mosaic image from a view point not seen in the input frames. So, at each location of the virtual camera, we need to select pixels from the input frames that are most fronto-parallel.



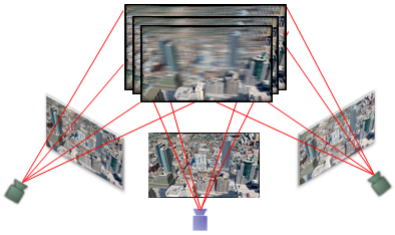
(a) Capturing an aerial video from a UAV Tour.



(b) Image registration and estimation of the camera motion trajectory (the blue curve).



(c) Choosing a reference camera by interpolating the source cameras.



(d) Generating the final mosaic depth map using plane sweep stereo and graph-cut optimization.

Figure 2: Illustration of the mosaic construction algorithm in [1]

A block diagram that illustrates the technique of [1] is given in figure 2 where a sequence of frames from a UAV video sequence (figure 2a) is used as an input to a Structure From Motion (SFM) algorithm [8] that will estimate the camera motion trajectory (figure 2b). The source cameras on the estimated trajectory are then interpolated (figure 2c) to define a reference camera with a wide FOV. Plane sweep stereo [2] then discretizes the 3D scene space into a set of sweep planes parallel to the reference camera and within a certain depth range. For each plane, The input frames are warped and averaged to define the plane image. Different scene objects will have correct focus at different depth levels (figure 2d), where their warped projections from the source images coincide well with their real 3D locations. So, we only need to locate the locations of good focus and merge them to define the output mosaic image on the image plane of the reference camera. In order to do that, a depth map is constructed by applying Graph-cut Optimization (GC) [9] on the intermediate depth planes and the final mosaic image is generated by copying pixels from the depth planes according to the generated depth map.

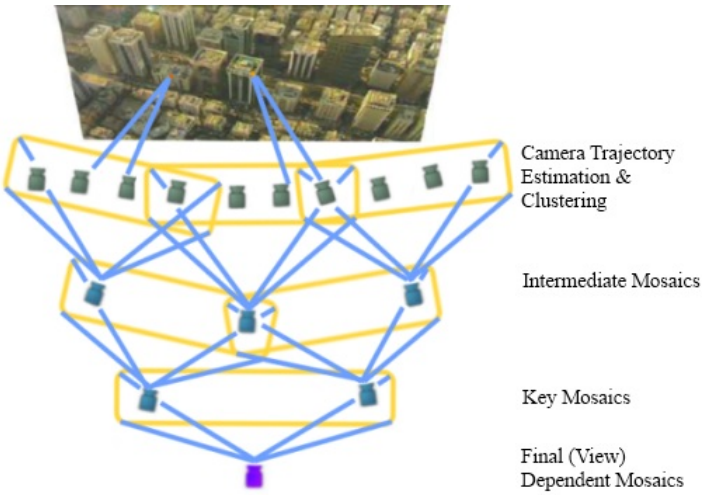


Figure 3: An overview of the presented mosaicing algorithm. SFM is used to estimate the camera parameters of each frame. The estimated cameras are then grouped into a set of overlapped clusters and plane sweep stereo is applied to generate intermediate mosaics. The intermediate mosaics are further grouped and a second phase of plane sweep stereo is used to define the key mosaics. The final view dependent mosaic is constructed by defining a virtual camera and stitching the intermediate and key mosaics onto this camera.

Although the technique of [1] can provide a good mosaic for a small sequence of video frames, It causes blurriness when applying it on a long sequences of video frames. This results from the increase of errors in the estimated camera parameters when projecting far camera locations to a reference camera. In order to solve this problem, the technique of [1] is extended into a new algorithm that consists of two stages, a data processing stage and a mosaic generation stage.

The data processing stage (Figure 3) clusters the input camera trajectory into a set of overlapped clusters and applies plane sweep stereo on each cluster to generate a depth map and a mosaic image. The set of generated mosaics are called intermediate mosaics and because there is a limit on the number of frames in each cluster, the mosaics will not suffer blurriness. A new camera trajectory is defined for the intermediate mosaics and a second phase of clustering and plane sweep stereo refines the depth maps and generates a second set of mosaics, called key mosaics.

The mosaic generation stage takes as input, the intermediate mosaics, the key mosaics and the pose of a virtual reference camera viewing the scene from a certain viewpoint. The intermediate and key mosaics are then projected on the image plane of the virtual camera and stitched together to generate a view dependent mosaic.

Plane sweep stereo is a very slow process if the depth range to sweep planes is very large and during the data processing stage, it is applied on each cluster

of a given long video sequence. This further increase the running time by large orders of magnitude. For example, if the algorithm performs θ amount of work to construct each depth plane then, we will do θn_d to construct n_d planes. We also add the amount of work γ to be carried by graph-cut to generate an interpolated mosaic to have $(\theta n_d + \gamma)$, and given n_c clusters, we will have $(\theta n_d + \gamma)n_c$. This is in the order of magnitude of $O(n^3)$ or much worse. This constitutes the most computational intensive part of our technique compared to the work performed in the mosaic generation stage to generate a view dependent mosaic. So, this project will focus on parallelizing the generation of intermediate and key mosaics using plane sweep stereo.

2 Problem Formulation

This section provides the mathematical definition of plane sweep stereo and illustrates the process of generating the intermediate and key mosaics from a clustered input trajectory. Because the project is focusing on the data processing stage, we will just give a highlight to the mosaic generation stage which constructs view dependent mosaics from the intermediate and key mosaics.

2.1 Plane Sweep Stereo

Let I be the set of input images and $P_i = K_i\,(R_i|t_i)$ is the camera for frame $I_i \in I, n_s = |I|$ where K_i represents the camera intrinsic parameters and $(R_i|t_i)$ are the camera rotation (related to a rotation vector $\mathbf{r}_i$) and translation components. We define an interpolated camera $P_v = K_v\,(R_v|t_v)$ with an interpolated image I_v where $K_v = K_i$ and $(R_v|t_v)$ are estimated as,

$$\mathbf{t}_v = \frac{1}{n_s}\sum_{i=1}^{n_s}\mathbf{t}_i \quad , \quad \mathbf{r} = \frac{1}{n_s}\sum_{i=1}^{n_s}\mathbf{r}_i \tag{1}$$

Given a signed distance d from the camera center $c_v = -R_v^T t_v$ normal to the image plane of P_v, there is only one plane n_d parallel to the image plane of P_v. If we assume that the 3D scene objects captured by the input cameras P_i are located on plane n_d, then the 2D points p_v of the image plane I_v are related to the 2D points p_i of a source image $I_i \in I$ via the plane transfer homography:

$$p_i = H_i(d)p_v \tag{2}$$

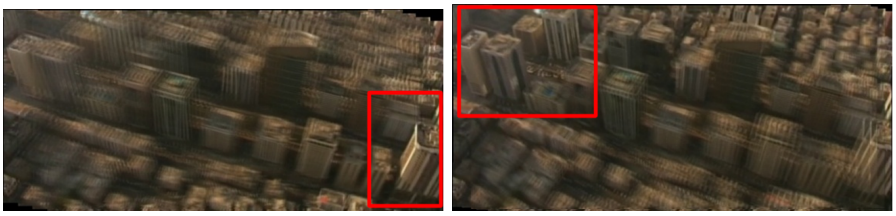
$$H_i(d) = K_i\left(R'_i - \frac{t'_i n^T}{d}\right)K_v^{-1} \tag{3}$$

where $n^T = (0, 0, -1)$, and R'_i, t'_i are the translation and rotation of camera P_i in the coordinate of the interpolated camera P_v and,

$$R'_i = R_i R_v^T \quad , \quad t'_i = -R_i R_v^T t_v + t_i \tag{4}$$

Usually scene objects will not lie in the same plane n_d unless the camera is very far from the scene that we can ignore the depth variations. In addition motion parallax can cause occlusion of objects so, objects that satisfy equation (2) and seen from camera P_v are the non-occluded objects. As we change the depth d new objects will become focused and others will appear blurred.

Figure 4 shows two sweep planes at depth $d = 30$ and $d = 60$ with different objects in focus. The pixels in the interpolated image I_v should be selected from sweep planes that has the correct focus or less blurring of pixels. This motivates the next step of plane sweep stereo that formulates a labeling problem to assign a depth to each pixel in I_v.



(a) Depth =17 (b) depth =26.5

Figure 4: Two different sweep planes that show different focus for scene objects.

In order to estimate the depth map, a depth label assignment problem [1, 3] $f : \Pi \rightarrow L$ is formulated that assigns a depth label from a set L (representing the depth planes) to each pixel in a set Π of the interpolated image I_v pixels. This problem is solved using multi-label graph-cut optimization [9] where the labeling energy is defined using a data cost term E_d and a smoothness cost term E_s,

$$E(f) = E_d(f) + E_s(f), \tag{5}$$

$$E_d(f) = \sum_{p \in I_v} A(p, f(p)), \tag{6}$$

$$E_s(f) = \sum_{\substack{p \in I_v \\ q \in N_p \\ d(p) \neq d(q)}} B(p, q, f(p), f(q)) \tag{7}$$

$A(p, f(p))$ is a function that penalizes the depth assignment $f(p)$ of p according to the color variance of the warped frames at p, and $B(p, q, f(p), f(q))$ encourages neighbouring pixels $p, q \in I_v$ to have similar depth labels $f(p), f(q)$. The penalty function A is defined as

$$A(p, d) = \alpha \sum_i (\bar{\rho}_{p,d} - \rho(C_{i,d}(p)))^2 \tag{8}$$

where $C_{i,d}(p)$ is the color of the pixel in the warped image I_i at depth d that corresponds to the pixel p of I_v using the homography (2). ρ is a summarization function and $\bar{\rho}_{p,d}$ is the average of color values of warped frames to plane d at

pixel location p. The function quadratically penalizes the colors that are away from the average color value. The summarization function ρ maps from the non-uniform RGB color space to the uniform HSV space. The smoothness function B is defined as,

$$B(p, q, d_p, d_q) = \beta \sum_i B_i(p, q, d_p, d_q) \quad (9)$$

$$B_i(p, q, d_p, d_q) = (d_p - d_q)^2 \quad (10)$$

This function encourages neighboring pixels in I_v to be selected from the same depth plane. The α and β parameters can be tweaked to control the tradeoff between smoothness and data energies.

After performing graph-cut optimization and recovering a depth map for the interpolated image I_v, we copy th color of each pixel p in I_v from the plane at the corresponding depth label $d(p)$. Pixels with large matching costs according to equation 5 indicate occluded regions and are left as holes. These holes are filled by scanning the interpolated image I_v row by row and marking the depth labels of the entry/exit pixels of each encountered hole and filling the in-between pixels from the farthest depth plane. This gives more weight to closer objects.

2.2 Construction of a view dependent mosaic

Figure 3 shows the steps of a mosaicing algorithm that constructs view dependent mosaics. The technique works in two stages the data processing stage and the mosaic generation stage. The data processing stage clusters the source camera trajectory into a set of overlapped clusters where each cluster contains a small number of successive frames. Plane sweep stereo [1] is then applied to recover a depth map and build a mosaic image for each cluster. The set of generated mosaics is called intermediate mosaics and are again clustered into a second group of overlapped clusters. Plane sweep stereo is again applied on each cluster to enhance the depth assignment and define another set of mosaics, called key mosaics. The key mosaics are similar to bridges between the intermediate mosaics to link and increase the smoothness.

The SFM algorithm provides a set of sparse 3D points detected from the input frames. These 3D points are projected to the intermediate and key mosaics where each 3D point has a 2D projection on a mosaic if the projection is within the boundary of that mosaic. This forms a set of keypoints on each mosaic which are used to stitch the mosaics from different viewpoints.

A virtual reference camera is defined at a certain location looking at the scene from a certain view point. The nearest mosaic to the virtual camera is selected and its keypoints are projected on the image plane of the virtual camera. The technique continues by projecting the keypoints of the other intermediate and key mosaics and aligns them to the same image plane.

After projecting all mosaics, a triangular mesh is generated using the aligned points such as the one shown in figure 5. Each triangle is filled from a certain

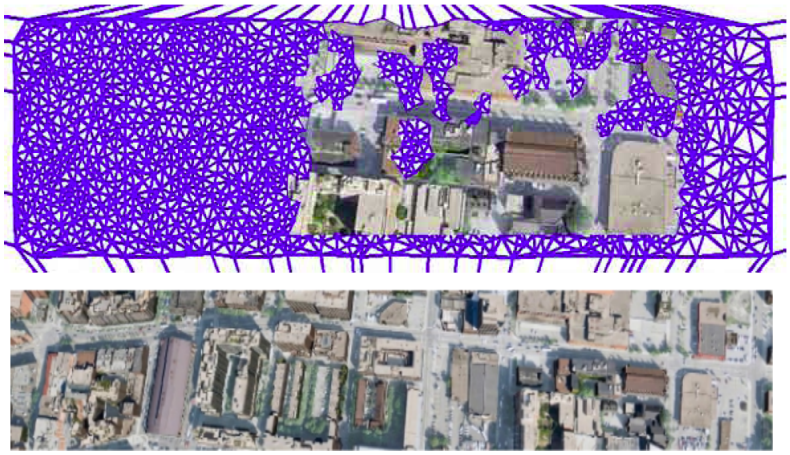


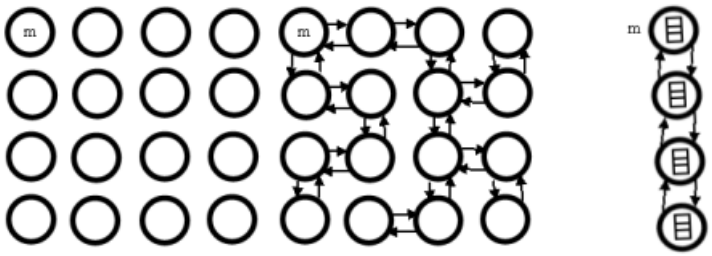
Figure 5: Stitching of the intermediate and key mosaics to generate a view specific mosaic. (Top row) Delaunay triangulation on the virtual camera image plane using the projected keypoints of intermediate and key mosaics. (Bottom row) An example of a constructed aerial mosaic.

mosaic based on a voting and shape matching scores. The voting score is determined for each triangle from the mosaics used to fill neighboring triangles. The shape matching score is calculated by comparing the shape of windows defined around the corners of each triangle to the corresponding locations in the mosaics that include this triangle. The final mosaic is the final image after filling all triangles.

2.3 The Parallel Algorithm

The parallel algorithm will focus on parallelizing the data processing stage using the Message Passing Interface (MPI). The goals set during the previous phase are,

- parallelizing the plane sweep stereo algorithm by assigning each process from a set of MPI processes, a number of depth planes to construct. Each process will perform frame warping and calculate the energy cost for each pixel on the given planes. A master process will then gather the calculated costs and the average plane images from all other processes and apply graph-cut optimization to generate the mosaic image.
- Given a clustered trajectory, we will need to apply the parallel plane sweep stereo on each cluster. This is performed by dividing the available MPI processes into subgroups where each group belongs to a communicator. Clusters will be grouped into a number of blocks and the resulted blocks will be distributed to the communicators. Each communicator will execute the parallel plane sweep stereo on the assigned block of clusters and collects



(a) Partitioning. (b) Communication. (c) Agglomeration and Mapping.

Figure 6: Applying foster's design methodology to parallelize the generation of sweep planes in PSS algorithm. (a) Partitioning the calculations of 16 sweep planes by associating one task for each plane. (b) We assume a communication layout where each task has at most $log(P)$ input and output channels for P number of processes. (c) Grouping the primitive tasks into different groups and mapping each group to one task. The letter m indicates the master process.

the resulted mosaics on a local master process. The local master processes from all communicators will then send the resulted mosaics to a global master process that will save these mosaics.

2.3.1 The Parallel Plane Sweep Stereo

The PSS algorithm is analyzed according to the foster's parallel algorithm design methodology [10]. Given a set of input cameras, the steps of the serial PSS are,

1. load the input frames and their associated camera parameters.
2. Calculate an interpolated camera.
3. Map the input cameras to the coordinates of the interpolated camera using equation 4.
4. Determine the depth range for plane sweeping.
5. Define the sweep planes within the depth range.
6. Calculate the transfer homographies of the input images to each sweep plane using equation 2.
7. Warp the input images to each sweep plane and calculate the average image of each plane.
8. Calculate the data costs using the color variance of equation 8.
9. Minimize equation 5 to recover an estimated depth map .

10. Recover the mosaic image of the interpolated camera by copying pixels from the average images of sweep planes according to the recovered depth map.

11. Apply hole filling for regions with large costs (above a certain threshold.)

Serial Algorithm Analysis: By analyzing the serial algorithm, we can identify that the steps from 2 to 6 can be calculated serially for a reasonable number of input images. Steps 7 and 8 can substantially increase the running time for a large depth range with many sweep planes. So, those steps must be parallelized for having a good performance. The energy minimization, step 9, is implemented by a 3rd party graph-cut library. So, it is performed serially on the data costs from all sweep planes. The last step is performed serially on the recovered depth map and constructs the mosaic image.

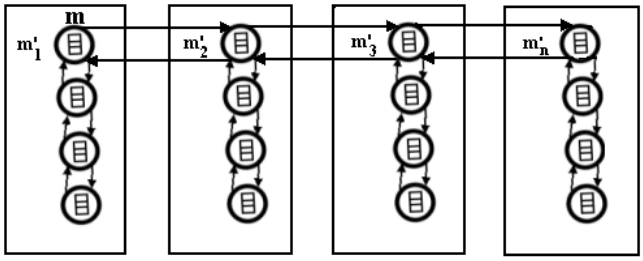
Parallel Algorithm Design: Figure 6 shows the design of the parallel PSS algorithm according to the foster's design method. Figure 6a shows a data-centric partitioning where a primitive task is associated with the calculations of each sweep plane used in PSS. Also, the communication layout (Figure 6b) is assumed to have $log(P)$ send/receive channels for each task where, P is the total number of tasks. This assumption is valid for several MPI collective communication functions. Usually, there will be a number of sweep planes larger than the number of processes. So, figure 6c shows the grouping of the primitive tasks into several blocks where each block has a number of sweep planes. The resulted blocks are then mapped to the available processes.

Parallel Algorithm Implementation: The implementation of the parallel PSS algorithm proceeds as follows: Given a set of MPI processes, a master process (marked with m in Figure 6) is responsible for loading the input frames and their associated camera parameters. The master process also performs the serial calculations of steps 2 to 6 of the PSS algorithm. Steps 7 and 8 are performed in parallel where, block decomposition is applied on the set of sweep planes to define several blocks. In order to calculate each plane, we need all the input frames and their associated transfer homographies. So, the master process should broadcast the input frames to all other processes and scatter the plane homographies to each process depending on the assigned group of planes. Each process will loop on its own set of sweep planes to calculate the average image and the data costs of each plane. The master process will then gather the plane images and data costs from all processes and proceeds serially with the remaining steps of the PSS algorithm to recover the output mosaic.

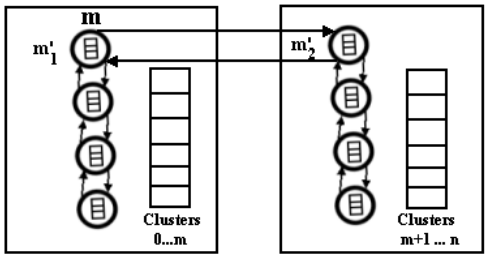
2.3.2 Parallelizing the data processing stage

The data processing stage is also analyzed according to the foster's design methodology. Given a long sequence of video frames, the serial steps of this stage are,

1. Load the input frames and the camera trajectory that contains the camera parameters of each frame.



(a) Partitioning.



(b) Agglomeration and Mapping.

Figure 7: Applying foster's design methodology to parallelize the data processing stage of the discussed algorithm. (a) Partitioning four overlapped clusters by associating an instance of the parallel PSS to each cluster. The parallel PSS instance includes tasks for calculating the sweep planes of each cluster with a local master task m'. (b) Grouping the clusters into different blocks and mapping each block to one parallel PSS instance. The capital letter **m** indicates the global master process.

2. Cluster the input frames into a set of overlapped clusters.
3. Apply the parallel PSS on each cluster to generate the intermediate mosaics.
4. Similar to step 2, Cluster the intermediate mosaics into a set of overlapped clusters.
5. Similar to step 3, Apply the parallel PSS on each cluster to generate the key mosaics.

Serial Algorithm Analysis: After analyzing these steps, we see that step 1 can be performed serially by one process to load the input frames and their associated camera parameters. Step 2 also, can be performed serially using a sliding window over the input frames. Step 3 needs to apply the parallel PSS algorithm on each generated cluster. Given a large set of clusters, this step will substantially decrease the performance and we need to perform it in parallel. The generated set of intermediate mosaics is forwarded to the remaining steps to generate the key mosaics. These steps are similar to the first three steps

and can be performed by re-applying the first three steps onto the intermediate mosaics. So, our implementation will only handle the first three steps.

Parallel Algorithm Design: Given a set of overlapped clusters, we need to apply an instance of the parallel PSS algorithm in figure 6c on each cluster and collect the generated mosaics from each instance to save them. This can be performed by splitting the set of available MPI processes into different instances of the parallel PSS algorithm.

Figure 7 shows the design of the data processing stage according to the foster's design method. Figure 7a shows a data-centric partitioning of the set of overlapped clusters by associating an instance of the parallel PSS to each cluster. Note that the parallel PSS instance contains a set of tasks controlled by a local master task m'. Usually, the number of clusters will be larger than the number of parallel PSS instances. So, figure 7b shows the grouping of the clusters into several blocks which are then mapped to the available parallel PSS instances.

Parallel Algorithm Implementation: In MPI, a communicator is an object that is created to ease the communication between a group of processes. MPI also, supports process grouping and allows the creation of virtual communicators by splitting the processes of a given communicator.

The parallel implementation makes use of the Split(color, key) function of the communicator object to create a set of virtual communicators. The Split function takes two arguments, the virtual communicator identifier or in MPI language, the color, and the key which controls the rank assignment of each process inside the created group. Each resulted communicator will have a certain number of processes that are shared with the parent communicator. A global master process is defined for the parent communicator and each group of processes will have a local master process. The user specifies the total number of processes and the maximum number of processes in each group.

After performing the process grouping, each group will be a parallel PSS instance that has a virtual communicator to handle all necessary communications. The global master process (indicated by **m** in figure 7) will perform the first two steps of the data processing stage to load and cluster the input frames and their associated camera parameters. The clusters are then grouped into different blocks based on the number of availabe parallel PSS instances. The global master process then sends each block to the local master process of the block's corresponding instance. The local master process loops on the clusters of the assigned block and apply the parallel PSS to generate a mosaic image for each cluster. When a local master process finishes the assigned block, it sends the generated mosaics back to the global master process which accumulates all the generated mosaics and save them to disk.

3 Results

The parallel implementation is tested on a synthetic aerial video sequence recorded from the Google Earth software package. The mosaic algorithm parameters are

(a) Sample input frames.

(b) Intermediate mosaic 1.

(c) Intermediate mosaic 2.

Figure 8: Generation of the intermediate mosaics. (Row 1) sample frames from the tested sequence. (Row 2) Examples of the generated intermediate mosaics.

as follows: the number of input frames is 16 and the size of each overlapped cluster is 6 frames. The MPI parameters are: the total number of processes is 20 and the number of processes in each process group is 4.

The parallel and serial algorithms are tested on the mako sharcnet cluster. Two experiments are performed with a difference only in the number of input frames. In the first experiment, the number of input frames is 8, The measured running time of the serial code is about 93 seconds and the measured running time of the parallel code is about 77 seconds. In the second experiment, the number of input frames is 16, The measured running time of the serial code is about 165 seconds and the measured running time of the parallel code is about 124 seconds. These results show the improved performance of the parallel algorithm toward the serial algorithm. Figure 8 shows sample input frames and the resulted intermediate mosaics constructed for some trajectory clusters.

References

[1] M. A. Helala, L. A. Zarrabeitia, and F. Z. Qureshi, "Mosaic of near ground uav videos under parallax effects," in *Proc. 6th ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC)*, Hong Kong, China, Oct. 2012.

[2] I. Geys, T. Koninckx, and L. Gool, "Fast interpolated cameras by combining a gpu based plane sweep with a max-flow regularisation algorithm," in *in Proc. 3DPVT, 2nd Int. Symp.*, Washington, DC, USA, Sept. 2004, pp. 534–541.

[3] Q. Zhi and J. Cooperstock, "Toward dynamic image mosaic generation with robustness to parallax," *IEEE Transactions on Image Processing*, vol. 21, no. 1, pp. 366–378, 2012.

[4] M. Quaritsch, R. Kuschnig, V. Mersheeva, D. Wischounig-Strucl, S. Yahyanejad, E. Yanmaz, G. Friedrich, H. Hellwagner, C. Bettstetter, and B. Rinner, "Col-

laborative uavs for aerial reconnaissance in rescue scenarios," in *Proc. Austrian Robotics Workshop*, Innsbruck, Austria, May 2011.

[5] D. Wischounig-Strucl, M. Quaritsch, and B. Rinner, "Prioritized data transmission in airborne camera networks for wide area surveillance and image mosaicking," in *Proc. IEEE CVPR*, vol. 1, Colorado Springs, USA, June 2011, pp. 692–698.

[6] D. Steedly, C. Pal, and S. Szeliski, "Efficiently registering video into panoramic mosaics," in *Proc. ICCV*, vol. 2, Beijing, China, Oct. 2005, pp. 1300–1307.

[7] R. Marzotto, A. Fusiello, and V. Murino, "High resolution video mosaicing with global alignment," in *Proc. IEEE CVPR*, vol. 1, Los Alamitos, CA, USA, June 2004, pp. 692–698.

[8] N. Snavely, S. Seitz, and R. Szeliski, "Modeling the world from internet photo collections," *International Journal of Computer Vision*, vol. 80, pp. 189–210, 2008, 10.1007/s11263-007-0107-3. [Online]. Available: http://dx.doi.org/10.1007/s11263-007-0107-3

[9] A. Delong, A. Osokin, H. N. Isack, and Y. Boykov, "Fast approximate energy minimization with label costs," *Int. J. Comput. Vision*, vol. 96, no. 1, pp. 1–27, Jan. 2012.

[10] M. J. Quinn, *Parallel Programming in C With Mpi and Openmp*. McGraw-Hill Higher Education, 2004.