Eurographics Symposium on Parallel Graphics and Visualization (2013)
F. Marton and K. Moreland (Editors)

Scalable Seams for Gigapixel Panoramas

S. Philip[1], B. Summa[1], J. Tierny[2], P.-T. Bremer[1,3] and V. Pascucci[1]

[1]Scientific Computing and Imaging Institute, University of Utah
[2]CNRS - Telecom ParisTech (LTCI)
[3]Lawrence Livermore National Labratory

Abstract

Gigapixel panoramas are an increasingly popular digital image application. They are often created as a mosaic of smaller images composited into a larger single image. The mosaic acquisition can occur over many hours causing the individual images to differ in exposure and lighting conditions. Therefore, to give the appearance of a single seamless image a blending operation is necessary. The quality of this blending depends on the magnitude of discontinuity along the boundaries between the images. Often image boundaries, or seams, are first computed to minimize this transition. Current techniques based on the multi-labeling Graph Cuts method are too slow and memory intensive for panoramas many gigapixels in size. In this paper we present a multithreaded out-of-core seam computing technique that is fast, has a small memory footprint, and gives near perfect scaling up to the number of physical cores of our test system. With this method the time required to compute image boundaries for gigapixel imagery improves from many hours (or even days) to just a few minutes on commodity hardware while still producing boundaries with energy that is on-par, if not better, than Graph Cuts.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—I.3.6 [Computer Graphics]: Methodology and Techniques—

1. Introduction

Panoramic images, composed as a mosaic of many smaller images, are an increasingly popular digital photography application. These images can range from a few megapixels to many gigapixels in size and can contain hundreds or thousands of individual images. Recently, the trend has been towards gigapixel sized panoramas due to the availability of high resolution cameras and inexpensive robots for automatic capture.

The robots capture the individual images one by one in a grid pattern and typically take a few seconds per image. Therefore, the capture of gigapixel sized panoramas can take many hours. This results in each image having different lighting and exposure conditions. The resulting panorama is an unappealing patchwork with visible transitions between the images. To combat this, blending operations are performed to give the appearance of a single seamless image. The quality of blending depends on the magnitude of discontinuity along the transition boundaries. Therefore, another step is usually performed where new boundaries between the images are computed such that the magnitude of transition between them is minimized. These boundaries are often called as *seams*.

Seam computation at its core is determining which source image provides the value for each pixel in regions where these images overlap. The most widely used technique for this problem is the Graph Cuts algorithm [BVZ01, BK04, KZ04]. This technique computes a k-labeling to the nodes of a graph in order to minimize an energy function defined on the nodes and edges of the graph. For the seam computation problem, the panorama is represented as a graph where the nodes represent the pixels, the edges connect a pixel to its neighbors, and labels represent the source images that provide the values for the corresponding pixels [KSE*03, ADA*04]. The energy function is typically pixel based and is used to minimize the color or color gradient variations between images. Graph Cuts is only an approximation since the globally optimal solution for more than two labels is known to be NP-hard [BVZ01].

Graph Cuts is not suitable for gigapixel sized images due to its high computational cost and memory requirements. A common technique employed to reduce both the memory and computational cost of Graph Cuts is to use a hierarchical scheme [LSGX05, AZP*05]. Hierarchical Graph Cuts has only been shown to produce good results for two to three levels [AZP*05] and can be seen often in practice as shown

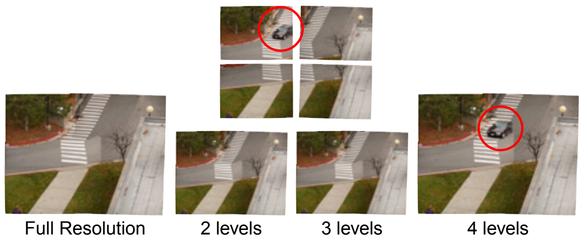


Figure 1: *An example case where a deep hierarchy for Hierarchical Graph Cuts produces poor results. A car that appears in only one of the input images (top) causes a region with high energy which a pixel based energy will avoid. The results for up to 3 levels are good. At 4 levels, in the coarse solution, the car is small enough for the seam to pass through it. The dilation parameter (10 pixels in the typical case) is not large enough to exit this local minima in the refinement stages.*

in Figure 1. In this figure, at 4 levels the dynamic scene element is small enough in the coarse solution for the seam to simply pass through it. The dilatation, which is the region around the upsampled coarse seams where refinement is performed, is not large enough (10 pixels is typical) to allow the seam to exit this local minima. Larger dilatation may help in this case although at an increased computational and memory cost. This interplay between a shallow hierarchy and dilatation parameters makes the technique inherently unscalable and not sufficient for direct application to gigapixel sized panoramas. A sliding window based out-of-core technique is presented in [KUDC07] where the Graph Cuts algorithm is applied over a window of each individual image of the panorama. This technique has been shown to work on gigapixel sized panoramas but it is sequential and is still computationally and memory intensive, even over the domains of individual images.

For faster seam computation, we look to the recently introduced alternative to Graph Cuts: Panorama Weaving [STP12]. This work presented a novel technique to combine independently computed pairwise seams into a global seam network. It has been shown to be fast, light-weight and easily parallelized. However, it is targeted towards interactive editing of panorama seams and as such is an in-core technique. Therefore, it has only been shown to work on panoramas of less than 100 megapixels.

In this work, we introduce a scalable version of the Panorama Weaving technique. In doing this, we present the first truly out-of-core, parallel and scalable panorama seam technique that can handle arbitrary sized panoramas yet requires limited memory independent of the total size of the panorama. It uses a caching technique to save redundant disk I/O. This cache is configurable to allow a user to trade memory overhead for reduced disk access.

2. Related Work

Once the images of a panorama are registered into a common global frame, it is desirable to smooth the transitions between the images to give the impression of a single seamless image. A simple approach is to perform an alpha-blend over the overlap areas. [Sze06] provides an introduction to this and other blending techniques. Although highly scalable, these techniques are not suitable for cases where there are registration errors, dynamic elements or varying lighting and exposure conditions across the images. These are very common for larger panoramas. Hard seams between the images that minimize the energy of transition can often hide the registration errors and dynamic elements and provide a good input for techniques such as gradient domain blending [PGB03, LZPW].

Panoramas where the images are acquired in a single sweep of the scene is a simplified case where only boundaries between pairwise images need to be considered [Mil75, Mil77, Sze96, SS98, Dav98, UES01]. The optimal boundary between a pair of overlapping images can be computed quickly and exactly using the min-cut/max-flow algorithm. There has been some work to combine such pairwise seams for more complex panorama configurations. [GMNG09] use an image distance based metric to combine the pairwise seams for more complex panoramas. [EF01] combine the seams by adding them together sequentially for the purpose of texture synthesis. [STP12] present a novel technique of combining these pairwise seams in a general panorama configuration by introducing the concept of a driving adjacency mesh data structure to encode the boundary relations and intersections in a panorama.

Graph Cuts builds on the min-cut/max-flow algorithm [BVZ01, BK04] to efficiently compute an approximate optimal solution for k-labeling of a graph. Graph Cuts has been adapted to the panorama seams problem [KSE*03, ADA*04] and it is currently a widely used technique. However, it is a computationally expensive and memory intensive technique and is not suitable for gigapixel sized images. As mentioned in the previous section, a hierarchical scheme can be used to alleviate some of these problems but it is inherently unscalable. The only truly scalable technique for panoramas is an out-of-core scheme where the Graph Cuts algorithm is applied over a window of each individual image [KUDC07]. As we will show in Section 5, this serial computation can take several hours to compute a solution.

There have been works that parallelize the min-cut/max-flow algorithm on multicore systems [LS10, JSH, DB08] and GPUs [VN08]. [DB08] is also capable of handling graphs that are larger than the available memory. However, min-cut/max-flow is not directly applicable to the panorama seams problem and extending it to multi-label Graph Cuts is non-trivial.

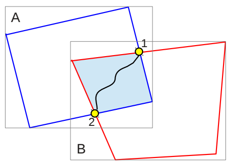


Figure 2: *Pairwise overlap between two images A and B. Their boundaries intersect at two points. The optimal pairwise seam is the min-path that connects these points.*

3. Panorama Weaving

In this section we give an overview of Panorama Weaving [STP12] which is the basis of our scalable seam computing technique.

Panorama Weaving produces a global seam solution by combining independently computed pairwise seams into the global seam network. Consider a pair of overlapping images as shown in Figure 2 with the boundaries of the images intersecting at two points. The pairwise seam between these images will be a path between the overlap pixels which connects these two intersection points and will partition the union of the images into two regions. This path can be computed by simply creating a dual graph where nodes are between pixels and the edges separate neighboring pixels. One can then compute the min-path on this graph, weighted with an energy function, using an algorithm such as Dijkstra's [Dij59]. The energy function is defined as $E_s(p,q)$ where $p,q \in \mathcal{N}$ and $\mathcal{N}$ is the set of all neighboring pixels. We would like to minimize the sum of the energy of all neighbors E with a labeling L. For the panorama boundary problem, this energy is typically [ADA*04] defined as:

$$E(L) = \sum_{p,q \in \mathcal{N}} E_s(p,q)$$

If minimizing the transition in pixel values:

$$E_s(p,q) = \|I_{L(p)}(p) - I_{L(q)}(p)\| + \|I_{L(p)}(q) - I_{L(q)}(q)\|$$

or if minimizing the transition in the gradient:

$$E_s(p,q) = \|\nabla I_{L(p)}(p) - \nabla I_{L(q)}(p)\| + \|\nabla I_{L(p)}(q) - \nabla I_{L(q)}(q)\|$$

where $L(p)$ and $L(q)$ are the labeling of the two pixels. The label of a pixel represents the source image that provides the value for that pixel. Notice that $L(p) = L(q)$ implies $E_s(p,q) = 0$.

To illustrate how the pairwise seams combine into a global seam network, consider a typical labeling of a panorama as shown in Figure 3(a). The labels form a collection of connected regions (shown in solid colors) that are separated by pairwise seams. A group of such seams meet each other at common points. These points are called the *branching points* of the seams. Figure 3(b), shows the pairwise seams and the branching points. Note that this is a simplified model, but as the previous work has shown this assumption still provides quality seam solutions.

The global seam network as described above can be represented by an abstract mesh data structure which is the dual

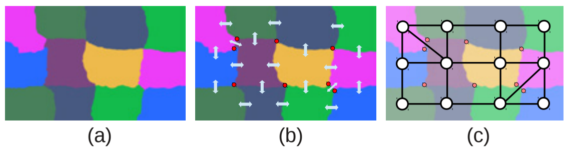


Figure 3: *(a) An example labeling of a panorama. (b) The labeling forms a set of connected regions separated by pairwise seams. Groups of seams join at points called as the Branching Points (shown in red). (c) A mesh representation of the panorama. The vertices correspond to the images, edges correspond to the pairwise seams and are orthogonal to them and the faces correspond to the branching points.*

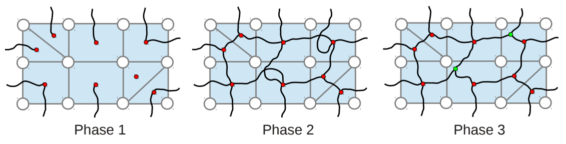


Figure 4: *The phases to compute the global seam network. Each face can be processed independently of each other in a phase. Phase 1 computes the branching points and boundary seams, phase 2 computes the shared seams and phase 3 detects and resolves seam intersections.*

of the seam network as shown in Figure 3(c). The vertices represent the images of the panorama, the edges represent the pairwise seams, to which they are are orthogonal, and the faces represent the branching-points. Given such a mesh representation of a panorama, its global seam network can be computed by processing the faces of the mesh independently of each other in logically three phases. Figure 4 gives an overview of these phases. In the first phase the branching points of the faces are computed. The seams on the boundary of the mesh can then be found, since their computation is independent of other faces. In the second phase the seams connecting the branching points are computed. These seams are each shared by two faces, but only one needs to perform the computation. The seams of a face can intersect and crossover each other causing areas of inconsistent labeling. In the third phase, intersections between seams of a face are detected and resolved.

The seams corresponding to the edges of a face meet at the face's branching point. Thus the branching point should be located in the region of intersection of all the pairwise overlaps represented by the edges, which is the same as the region of intersection of all the images corresponding to the vertices. Therefore, the images of a face must overlap each other. In short, the vertices of a face form a clique of overlap relationship. The intersection region is called as the face's *Multi Overlap* region. Figure 5 shows an example configuration for a quad face. Only the pairwise overlaps represented by the edges of the face need to be considered. For each overlap, the intersection point that is closest to the multi-overlap region is labeled as the *inside* point and the other one

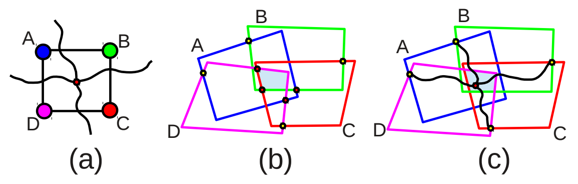


Figure 5: *(a) A quad face of a panorama mesh with its branching point and seams. (b) The collection of images that form the face. These images have a non-empty intersection called as the Multi-Overlap region. Only the pairwise overlaps corresponding to the edges need to be considered. The intersection point of an overlap closer to the multi-overlap region is the inside point (red) and the other is the outside point (yellow). (c) The inside points are adapted into a branching point. The branching point is located inside the multi-overlap and minimizes the min-path distances from the outside points.*

is labeled as the *outside* point (Figure 5(b)). The seams are combined by replacing the inside points with the branching point. A single-source/all-destinations min-path tree, called the *Seam Tree*, is computed on the dual graph of each overlap with the corresponding outside point as the source. Each node of a seam-tree gives the cost of the path from the source to that node. Within the multi-overlap region, the node for which the sum of its costs from each of the seam-trees is minimized, gives a good location for the branching point.

For faces that contain the mesh-boundary edges, seams that are orthogonal to those edges are computed by a simple lookup in the edge's seam-tree from the face's branching point. Once the branching points for all the faces are computed, seams orthogonal to the interior edges of the mesh are computed by connecting the branching points of its adjacent faces with a min-path on the energy field defined over its overlap.

There is a possibility that the paths of the individual seams of a face will intersect. The details about how and why the seams intersect, its implications and resolution are beyond the scope of this paper. Interested readers are referred to [STP12] for further details. At a high level, intersections between pairs of seams are handled by truncating the seams up to the furthest intersection point and computing a new seam path to the intersection point over a proper energy function.

The mesh representation of a panorama can be generated from the collection of its input images by creating an adjacency graph where the nodes represent the images and the edges represent each pairwise overlap between the images. All non-overlapping maximal cliques of this graph are identified and the edges that form the boundary of the cliques are activated. The cliques then become the faces and the active edges become the edges of the mesh.

4. Scalable Seams

Panorama Weaving is an in-core technique that needs all the images and intermediate buffers, such as overlap energies and seam trees, in memory. This is not scalable to gigapixel panoramas as the memory usage greatly increases with the total number of images. In this section, we describe our scalable seams technique.

4.1. Out of Core Seams

One of the strengths of Panorama Weaving is that the faces of the mesh representation can be processed independently. We use this feature to process one face at a time and only use the memory required per face. Moreover, the processing within a face is ordered such that all the images and buffers of the face need not be active at all times and can be acquired and released as needed. There is a trade off between speed and memory as acquiring and releasing results in increased disk I/O and/or recomputation of buffers. Therefore, it is necessary to reduce the number of times a particular buffer is acquired for good performance. We use a caching scheme for this purpose. The size of this cache can be configured based on the amount of memory available in a particular system and/or a user's preference. The rest of the section describes the Scalable Seams technique in detail:

Input. The input is typically a collection of images that have already been registered, transformed and rasterized along with their bounding boxes in the panorama. The invalid pixels of the images have an alpha value of zero and this property is used to identify the shape of the image.

Preprocessing. A pair of images is assumed to be overlapping if their bounding boxes overlap. This information is used to build the full adjacency graph of the images. The mesh representation is then computed from this adjacency graph as described in section 3. In case of gigapixel panoramas, since the images are typically acquired by robots on a grid a simplifying assumption can be made that the panorama forms a quad mesh layout.

Pass 1. The edges of a face are assigned a winding order (Figure 6 top) and we process the edges in that order. Figure 6 shows the various data buffers required for the computation of the branching point, and their dependencies. For each edge, the images corresponding to their vertices are loaded and their overlap energy is computed. The image buffers can now be released. Next, the seam tree is computed after which the energy buffer can be released. The costs of the nodes of the tree in the multi-overlap region are accumulated in a cost buffer, at which point the seam tree can be released. After the costs from all seam trees of the face have been accumulated, the location of the minimal value in the cost buffer will be the location of the branching point. Another iteration through the edges is performed and for mesh-boundary edges, the corresponding seam path can be found using the edge's seam tree. A cost-memory trade-off can be

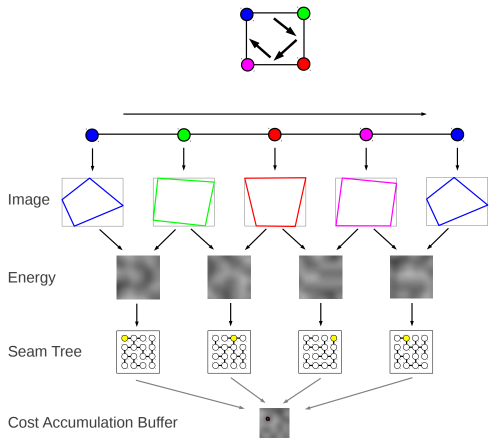


Figure 6: *Computation of the branching point of a face and the dependencies between the required data structures. The edges are iterated over based on a winding order (top) The images corresponding to the two vertices are loaded and the overlap energy is computed. A single-source/all-destinations min-path tree (seam-tree) is computed on the energy. The costs to its nodes within the multi-overlap region are accumulated into an accumulation buffer. The minimal point in this buffer gives the location of the branching point. Buffers can be freed once their dependencies have been computed. Edges are processed sequentially so only one set of these buffers need to be active at a time.*

made at this point, if seam trees of these edges are not freed in the first iteration. In this way the reloading of images and recomputation of the overlap energy and seam-tree can be avoided. Another such trade-off can be made for image loading. Since the second image of an edge becomes the first of the next edge, reloading of the image can be avoided by maintaining a circular buffer of length two for the images. The first vertex of the first edge is also the second vertex of the last edge. By keeping the first image around for the entire iteration on the edges the number of times that each image of a face is loaded can be reduced to just once.

Pass 2. For all shared edges the corresponding seam is computed as a min-path on the overlap energy between the branching points of the adjacent faces. Note that passes 1 and 2 do not strictly need to be two separate passes. During the first pass, if the branching-point for an adjacent face is available, a face can simply compute the proper seam path after its branching point is found. This can be checked by having a counter for each shared seam that counts the number of branching points that have been computed. For parallel implementations this counter should be atomic. After computation of a branching point the counter is incremented for each of the face's shared seam. The face for which the counter becomes two is the face that computes the seam. Merging the two passes also allows us to save the overlap energy (computed during the first iteration over the edges) of the shared edges and reuse that for the min-path computation.

Pass 3. Pairwise seams of a face are checked for intersections. If detected, the furthest intersection point is identified and the seams are truncated to this point. Note that seams that are shared can have intersections in both of its faces. To be able to process this pass for the faces independently, the seam paths are implemented as double ended queues. To truncate a shared seam one face only needs to update its head and the other only updates its tail with no conflict between the two. A new seam is then computed from the branching point to the intersection point. The required images and overlap energies are loaded and computed as required.

Output. The output of our system is a set of seams. Each seam is represented by the labels they separate the region into and their paths. The region of each image in the panorama can be rasterized from the seams that have the image as one of their labels.

4.2. Implementation Details

For testing, we have implemented four different versions of our technique. First a single threaded, out-of-core version that sequentially iterates over each individual face. Two passes are performed, to compute the seams and resolve the seam intersections. The computation of branching points, shared seams, and boundary seams are merged into a single pass as described previously. The intersection resolution is still performed in a separate pass. We chose to implement the cost-memory trade-offs mentioned in the previous subsection to achieve a balanced memory usage vs performance.

The second version is a *direct* parallelization of the sequential out-of-core version. A pool of threads is maintained and each thread processes one face at a time. After processing a face, a thread chooses the next unprocessed face from a global work queue. Each face is put into the work queue twice - once for branching point and seams computation and again for intersection resolution. In this version, there is no separate pass for intersection resolution. As soon as all the seams of a particular face have been computed it is put back in the work queue for intersection resolution. This way we avoid a stall in the pipeline that would have resulted from separate passes.

The third version is an extension of the direct parallel method. Note that nodes and edges are shared between neighboring faces. This corresponds to sharing of images and pairwise overlaps. Due to dynamic scheduling, it is likely that parallel threads are working on adjacent faces. Given that the images and buffers are read-only data, we can reduce I/O and computation time by sharing these resources among the threads. The image and overlap buffers are tracked by reference counts, whenever a thread requires a resource it increments its reference count and decrements

Figure 7: *Seams computed on the SLC panorama - 3.27 gigapixels with 624 individual images. The results were generated in 3.76 minutes using 24 threads and 3.0 GB of memory.*

Figure 8: *Seams computed on the Lake Louise panorama - 7.52 gigapixels with 1512 individual images. The results were generated in 14.71 minutes using 24 threads and 4.7 GB of memory. Panorama courtesy of City Escapes Nature Photography.*

it when it is done. The buffers are only freed when its reference count goes to zero. We call this the *sharing* version.

The fourth is the *caching* version which further improves upon the previous version by not immediately freeing the resources whose reference count has reached zero. Instead it can be configured with a user defined cache size. When the memory usage starts exceeding this size, a cache manager starts freeing buffers with reference count zero in a *Least Recently Used* manner until the memory usage returns to the set threshold. This allows the implementation to use more memory in systems that can afford it to further reduce the disk I/O and recomputation.

5. Results

We have conducted tests to compare our technique with the moving window implementation of Graph Cuts [KUDC07] with additional strong scaling tests of our technique. The following two datasets were used for testing:

- **SLC.** $122,625 \times 26632$, 3.27 gigapixels panorama composed of 624 individual images acquired in a 48×13 grid. Figure 7.
- **Lake Louise.** $187,068 \times 40201$, 7.52 gigapixels panorama composed of 1512 individual images acquired in a 72×21 grid. Figure 8.

5.1. Comparison with Graph Cuts

As mentioned in Section 2, the moving window application of Graph Cuts [KUDC07] is the only work that has been shown to work with gigapixel sized panoramas and therefore is used for our comparison. We also extended the implementation by adding support for hierarchical solving. The implementation sequentially applies the Graph Cuts algorithm over the window of each individual image. For each

SLC			
	Time (min.)	Max MB	Total Energy
GC1	1533.97	3324	1.070×10^8
GC3	318.21	867	1.119×10^8
SS	64.6	290	1.036×10^8

Lake Louise			
	Time (min.)	Max MB	Total Energy
GC1	8037.82	4022	2.927×10^8
GC3	1029.88	1067	3.279×10^8
SS	266.64	382	2.841×10^8

Figure 9: *Results of our single threaded out-of-core implementation (SS) and the two sliding-window based out-of-core Graph Cuts implementations - Full resolution version (GC1) and Hierarchical version with 3 levels (GC3). GC1 is too slow to be practical. GC3 is faster and uses lesser memory but produces poorer results. Our method is much faster than GC3 and uses much less memory while producing seams that are better than even GC1.*

image, it and its overlapping images are loaded and the energy function is computed over the domain of the image. The Graph Cuts algorithm is then applied on the energy for the solution of the window. Overlapping portions from solutions of previous windows are locked so that the seams are consistent across window boundaries. For hierarchical implementation an image pyramid with the specified number of levels is created per each window. Graph Cuts is first applied to the energy computed at the deepest level. The solution is then upsampled to the next level using bilateral upsampling. A dilation is applied on the upsampled seams and Graph Cuts is again performed on the dilated region. This step is repeated till the top level of the pyramid. For Graph Cuts we use the widely used implementation provided by [BVZ01, BK04, KZ04, DOIB10].

We tested the Graph Cuts based implementation on our datasets with two different configurations - one running the solver at full resolution and the other with 3 levels of hierarchy. We compared the running times, memory usage and total energy of the results with the single threaded, out-of-core implementation of our technique. The tests were run on a system with a quad core Intel Core i7 920 CPU @ 2.67 GHz and 6 GB of memory.

The results are detailed in table 9. Even though the out-of-core Graph Cuts implementation can handle gigapixel sized panoramas, it is too slow for practical purposes. The hierarchical solver is much faster and requires lesser memory at the cost of a lower quality solution. In contrast, our system is much faster, uses much less memory and the final energy computed is lower than the full resolution Graph Cuts implementation.

5.2. Scalability Tests

We have performed scalability tests on our three parallel, out-of-core implementations. The system for these tests was an Intel Xeon based workstation with 4 Intel Xeon E7540 CPUs @ 2.00 GHz, having 6 physical cores and 12 HT threads each and 128 GB of RAM. Table 10 shows the performance results for SLC and Lake Louise datasets.

The *direct* threaded implementation is fast and scalable and exhibits near linear scaling throughout. Even for up to the maximum number of physical cores the efficiency is maintained around 95% for SLC dataset. The slight reduction in efficiency at 24 threads is due to the smaller size of the panorama and it is not seen in the larger Lake Louise dataset, which is maintained at 99% efficiency. From the table it can be seen that the maximum memory usage increases with the number of threads for the direct and shared implementations, but memory/thread is always maintained bellow the usage of single threaded runs.

The *sharing* implementation shows an overall improvement in the performance compared to the *direct* implementation. By sharing the buffers among the threads it reduces their recomputation and disk I/O. This results in a super-linear speed-up as shown in the graph with the efficiency reaching as high as 114% for SLC and 117% for Lake Louise datasets. The data sharing also reduces the maximum memory usage of this implementation. As shown in the table, for SLC dataset it only requires 3.0 GB of memory with 24 threads compared to 4.0 GB for the *direct* implementation. Similarly for Lake Louise dataset it only requires 4.8 GB of memory with 24 threads compared to 5.8 GB for *direct*.

For the *caching* implementation, the tests were configured to use 8 GB of cache which is a reasonable size for modern workstations. With this implementation we are able to further improve the performance while still maintaining its scalability. As the table shows, the efficiency doesn't go bellow 94% for SLC and 98% for Lake Louise datasets. The maximum memory usage is configured to 8 GB.

6. Conclusion

In this work, we have presented a technique for computing seams for panoramas that is fast, light and scalable. On resource constrained system it is able to run in an out-of-core fashion using very little memory and still produce a solution orders of magnitude faster than the previous state-of-the-art: a moving window Graph Cuts scheme. On multi-core systems it can run in parallel and achieve super-linear speed-ups by sharing data among the threads and reduce I/O and computation. On systems with higher memory resources it can use caching to further reduce I/O and improve the performance. Moreover the energy of the seams produced by our technique is comparable to the previous work and in most cases, better.

Panorama Weaving allows for interactive editing of seams. For future work, we plan to extend our technique to allow local editing of seams for large panoramas in a scalable and interactive manner.

Acknowledgements

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

[ADA*04] Agarwala A., Dontcheva M., Agrawala M., Drucker S. M., Colburn A., Curless B., Salesin D., Cohen M. F.: Interactive digital photomontage. *ACM Trans. Graph 23*, 3 (2004), 294–302. 1, 2, 3

[AZP*05] Agarwala A., Zheng K. C., Pal C., Agrawala M., Cohen M. F., Curless B., Salesin D., Szeliski R.: Panoramic video textures. *ACM TOG 24*, 3 (2005), 821–827. 1

[BK04] Boykov Y., Kolmogorov V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Anal. Mach. Intell 26*, 9 (2004), 1124–1137. 1, 2, 6

[BVZ01] Boykov Y. Y., Veksler O., Zabih R.: Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Analysis and Machine Intelligence 23*, 11 (Nov. 2001). 1, 2, 6

[Dav98] Davis J. E.: Mosaics of scenes with moving objects. In *CVPR* (1998), pp. 354–360. 2

[DB08] Delong A., Boykov Y.: A scalable graph-cut algorithm for N-D grids. In *CVPR* (2008), IEEE. 2

[Dij59] Dijkstra E. W.: A note on two problems in connexion with graphs. *Numerische Mathematik 1* (1959), 269–271. 3

[DOIB10] Delong A., Osokin A., Isack H., Boykov Y.: Fast approximate energy minimization with label costs. In *2010 IEEE Conference on CVPR* (june 2010), pp. 2173 –2180. 6

[EF01] Efros A. A., Freeman W. T.: Image quilting for texture synthesis and transfer. In *SIGGRAPH* (2001). 2

[GMNG09] Gracias N. R. E., Mahoor M. H., Negahdaripour S., Gleason A. C. R.: Fast image blending using watersheds and graph cuts. *Image and Vision Computing 27*, 5 (Apr. 2009), 597–607. 2

[JSH] Jamriska O., Sykora D., Hornung A.: Cache-efficient graph cuts on structured grids. In *2012 CVPR*. 2

[KSE*03] Kwatra V., Schödl A., Essa I., Turk G., Bobick A.: Graphcut textures: Image and video synthesis using graph cuts. *ACM Trans. Graph 22*, 3 (July 2003), 277–286. 1, 2

[KUDC07] Kopf J., Uyttendaele M., Deussen O., Cohen M. F.: Capturing and viewing gigapixel images. *ACM Trans. Graph 26*, 3 (2007), 93. 2, 6

[KZ04] Kolmogorov V., Zabih R.: What energy functions can be minimized via graph cuts? *IEEE Trans. Pattern Anal. Mach. Intell 26*, 2 (2004), 147–159. 1, 6

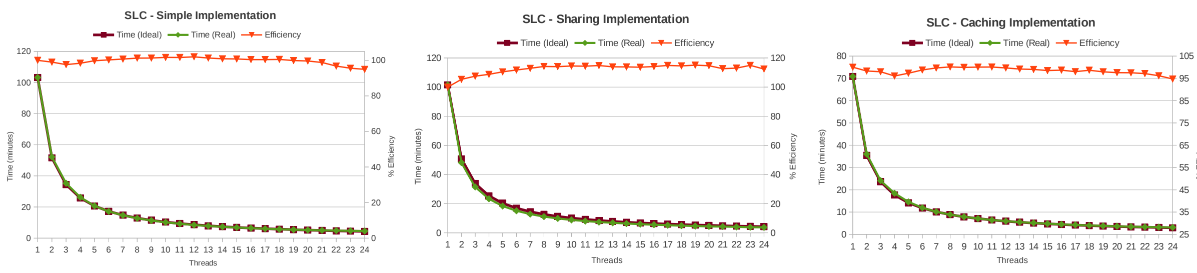
[LS10] Liu J., Sun J.: Parallel graph-cuts by adaptive bottom-up merging. In *CVPR* (2010), IEEE, pp. 2181–2188. 2

[LSGX05] Lombaert H., Sun Y. Y., Grady L., Xu C. Y.: A multilevel banded graph cuts method for fast image segmentation. In *ICCV* (2005), pp. I: 259–265. 1

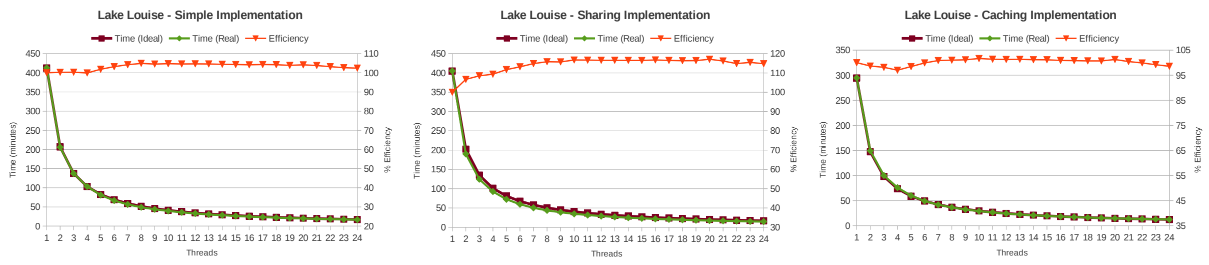
[LZPW] Levin A., Zomet A., Peleg S., Weiss Y.: Seamless image stitching in the gradient domain. In *ECCV 2004*. 2

[Mil75] Milgram D. L.: Computer methods for creating photomosaics. *IEEE Trans. Computer 23* (1975), 1113–1119. 2

[Mil77] Milgram D. L.: Adaptive techniques for photomosaicking. *IEEE Trans. Computer 26* (1977), 1175–1180. 2



SLC												
	Direct				Shared				Cached (8 GB)			
Threads	Ideal	Actual	Eff. %	MB	Ideal	Actual	Eff. %	MB	Ideal	Actual	Eff. %	MB
1	103.26	103.26	100.00	290	101.53	101.53	100.00	290	70.86	70.86	100.00	8294
4	25.82	26.23	98.41	784	25.38	23.34	108.75	769	17.72	18.44	96.08	8323
8	12.91	12.75	101.26	1412	12.69	11.12	114.17	1306	8.86	8.85	100.13	8323
12	8.61	8.44	101.99	2207	8.46	7.37	114.79	1869	5.91	5.92	99.70	8383
16	6.45	6.43	100.43	2636	6.35	5.56	114.10	2179	4.43	4.49	98.69	8386
20	5.16	5.18	99.63	3280	5.08	4.43	114.70	2624	3.54	3.63	97.56	8403
24	4.30	4.53	94.93	4018	4.23	3.76	112.41	2991	2.95	3.12	94.72	8415



Lake Louise												
	Direct				Shared				Cached (8 GB)			
Threads	Ideal	Actual	Eff. %	MB	Ideal	Actual	Eff. %	MB	Ideal	Actual	Eff. %	MB
1	412.85	412.85	100.00	382	405.10	405.10	100.00	382	294.62	294.62	100.00	8304
4	103.21	103.26	99.95	1129	101.27	92.60	109.37	1052	73.65	75.92	97.01	8393
8	51.61	49.19	104.91	2099	50.64	43.73	115.79	1891	36.83	36.46	101.01	8413
12	34.40	32.86	104.69	3079	33.76	28.97	116.52	2536	24.55	24.24	101.27	8481
16	25.80	24.77	104.16	3953	25.32	21.69	116.71	3471	18.41	18.25	100.90	8526
20	20.64	19.82	104.17	4860	20.25	17.30	117.07	4395	14.73	14.55	101.28	8861
24	17.20	16.79	102.46	5820	16.88	14.71	114.76	4750	12.28	12.45	98.58	8599

Figure 10: *Scaling results of our three parallel implementations on the SLC (top) and Lake Louise (bottom) datasets. Due to space limitations the tables only show numbers for core counts in multiples of 4. The plots show all the data points. (All timings are in minutes). All implementations show good efficiency (Eff. %) throughout, with the shared implementation achieving super-linear speedup due to data sharing. The maximum memory usage (MB) of the direct and shared implementations do increase with the number of threads but memory/thread is always maintained bellow the usage of single threaded runs. The cached implementation can be configured with a cache size (8 GB in this case) that allows it to scale to systems with larger memory. It provides the best performance among all three implementations by further reducing disk IO and computations.*

[PGB03] PÉREZ P., GANGNET M., BLAKE A.: Poisson image editing. *ACM Trans. Graph 22*, 3 (2003), 313–318. 2

[SS98] SHUM H. Y., SZELISKI R. S.: Construction and refinement of panoramic mosaics with global and local alignment. In *ICCV* (1998), pp. 953–956. 2

[STP12] SUMMA B., TIERNY J., PASCUCCI V.: Panorama weaving: fast and flexible seam processing. *ACM Trans. Graph. 31*, 4 (July 2012), 83:1–83:11. 2, 3, 4

[Sze96] SZELISKI R. S.: Video mosaics for virtual environments. *IEEE Computer Graphics and Applications 16*, 2 (Mar. 1996). 2

[Sze06] SZELISKI R.: Image alignment and stitching: A tutorial. *Foundations and Trends in Computer Graphics and Vision 2*, 1 (2006). 2

[UES01] UYTTENDAELE M. T., EDEN A., SZELISKI R. S.: Eliminating ghosting and exposure artifacts in image mosaics. II:509–516. 2

[VN08] VINEET V., NARAYANAN P. J.: CUDA cuts: Fast graph cuts on the GPU. In *Computer Vision on GPU* (2008), pp. 1–8. 2