Efficient Gradient-Domain Compositing Using Quadtrees

Aseem Agarwala
Adobe Systems, Inc.

Abstract

We describe a hierarchical approach to improving the efficiency of *gradient-domain compositing*, a technique that constructs seamless composites by combining the gradients of images into a vector field that is then integrated to form a composite. While gradient-domain compositing is powerful and widely used, it suffers from poor scalability. Computing an n pixel composite requires solving a linear system with n variables; solving such a large system quickly overwhelms the main memory of a standard computer when performed for multi-megapixel composites, which are common in practice. In this paper we show how to perform gradient-domain compositing approximately by solving an $O(p)$ linear system, where p is the total length of the seams between image regions in the composite; for typical cases, p is $O(\sqrt{n})$. We achieve this reduction by transforming the problem into a space where much of the solution is smooth, and then utilize the pattern of this smoothness to adaptively subdivide the problem domain using quadtrees. We demonstrate the merits of our approach by performing panoramic stitching and image region copy-and-paste in significantly reduced time and memory while achieving visually identical results.

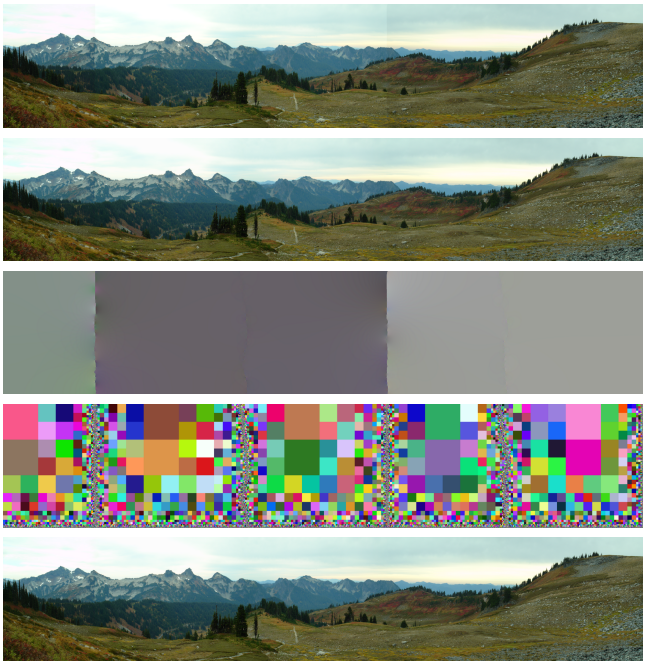

Figure 1 A 17-megapixel panorama shot with a hand-held camera. (First row) Panorama created by simply copying colors from the source images. Notice the subtle vertical seams caused by variations in exposure. (Second row) The gradient-domain composite. (Third row) A visualization of the difference between the first two images. The key observation of our work is that this difference exhibits intricate detail near the seams between image regions, but becomes progressively smoother away from these seams. (Fourth row) To take advantage of this smoothness, we subdivide the domain using a quadtree such that maximum subdivision occurs along the seams. (Fifth row) The result computed in this reduced space, which can be computed much more efficiently, is visually identical to the full gradient-domain solution. The numerical error is shown in Table 1. Images courtesy of Tobias Oberlies.

1 Introduction

Many recent algorithms for combining regions of multiple photographs or videos into a seamless composite operate in the gradient domain. Rather than copying absolute colors from the source images into a composite, these algorithms instead copy color gradients between source pixels and their immediate neighbors to form a composite vector field. A composite image (or video) whose gradients best match this composite vector field in a least squares sense is then reconstructed by solving a linear system (equivalent to the discretized Poisson equation) whose variables are the colors of each pixel [Pérez et al. 2003].

This technique is one of the most widely used algorithms in computational photography and video; unfortunately, however, it does not scale well to the multi-megapixel digital imagery common today. Solving a linear system on the order of the number of pixels quickly becomes prohibitive both in terms of time and space. Thus, despite the broad applicability of gradient-domain techniques, this poor scalability has limited their adoption in digital photography software. [1]

In this paper, we describe a simple and novel approach to gradient-domain compositing that greatly reduces the scale of the problem. We show how to approximately compute an n pixel gradient-domain composite by solving a linear system whose number of variables is $O(p)$ rather than n, where p is the total length of the seams between the image regions in the composite. This length will be much smaller than n, and can be shown to be $O(\sqrt{n})$ for typical cases. Solving this reduced system greatly reduces time and memory requirements yet achieves results that are visually identical. We achieve this efficiency increase by observing that the difference between a simple color composite and its associated gradient-domain composite is largely smooth (Figure 1), and the pattern of this smoothness can be predicted *a priori*. We thus solve for this difference, and adaptively subdivide the domain using a *quadtree* (a hierarchical spatial data structure [Samet 1990]) so that smoother areas of the solution are interpolated using fewer variables.

Efficient algorithms for solving the Poisson equation such as multigrid methods [Saad 2003] are well-studied and can be adapted to the GPU [Bolz et al. 2003]. Szeliski [2006] recently introduced a preconditioner that greatly accelerates the convergence of an iterative

`http://agarwala.org/efficient_gdc/`

[1] An exception is Adobe Photoshop's Healing Brush [Georgiev 2004], which is efficient because it operates on only small regions of an image at any one time.

ACM Reference Format
Agarwala, A. 2007. Efficient Gradient-Domain Compositing Using Quadtrees. *ACM Trans. Graph. 26*, 3, Article 94 (July 2007), 5 pages. DOI = 10.1145/1239451.1239545 http://doi.acm.org/10.1145/1239451.1239545.

conjugate gradient solver. These techniques, however, do nothing to address the fundamental *scale* of the problem; they still require solving linear systems on the order of the number of pixels. Thus, even though the number of iterations required to solve the linear system may be greatly reduced, the $O(n)$ memory required by a sparse solver can still quickly overwhelm the capabilities of a typical computer. Worse, iterative solvers do not exhibit the data reuse patterns that would make them well-suited to an efficient out-of-core implementation [Toledo 1999], since iterations computed only within a local area of the domain may not make significant progress towards the correct, global solution. In contrast, by solving an $O(p)$ variable linear system we can compute very large gradient-domain composites in surprisingly little memory (Table 2). The main memory used by our approach is $O(p)$ (since the input and output images are streamed through the algorithm tile-by-tile), but asymptotically the execution time is still $O(n)$ since each input and output pixel must be visited. However, those stages of our algorithm that are $O(n)$ are not time-consuming, and the solution of the $O(p)$ linear system remains the bottleneck.

We are not the first to adaptively vary resolution when solving linear systems or discretized partial differential equations. Losasso et al. [2004] performed large-scale fluid simulations by solving the Poisson equation on octree grids, and Szeliski and Shum [1996] used quadtrees for hierarchical motion estimation in video. To the best of our knowledge, however, we are the first to show how quadtrees can be applied to efficient gradient-domain compositing, and the transformations of the problem necessary to make this approach effective. We are also the first to demonstrate very large gradient-domain composites computed in reasonable time and, more importantly, space.

2 Gradient-domain compositing

Gradient-domain compositing hides seams between composited image regions by converting high-frequency artifacts that may appear at the boundaries between composited regions into low-frequency variations that spread across the image. This approach is effective because the human visual system is much more sensitive to local contrast than to slow changes in luminance and chrominance [Palmer 1999]. Gradient-domain compositing takes advantage of *lightness constancy* — our ability to discount the effects of scene illumination in order to perceive the true reflectivity of a scene. Retinex theory [Land 1977] suggests that humans achieve lightness constancy by perceiving scene lightness only through local luminance ratios at edges; low-frequency variations in luminance are discounted as the effects of illumination.

The usefulness of combining image regions in the gradient-domain was first described by Perez et al. [2003]; they demonstrated the ability to seamlessly copy a region from one image into another, as well as a variety of other image editing operations. Georgiev [2004] revealed that the Adobe Photoshop Healing Brush uses a similar technique, and Jia et al. [2006] improved on this basic approach by first optimizing the boundary of the copied region. Agarwala et al. [2004] extended gradient-domain compositing to the case of compositing an entire image from regions of many sources, within the context of a general, interactive framework for combining sets of images into a photomontage. This system used both graph cuts [Kwatra et al. 2003] for optimal seam selection and gradient-domain compositing for removing any artifacts remaining at the seams. This approach has been used to fill holes in images [Hays and Efros 2007] and to compute multi-viewpoint [Agarwala et al. 2006] and video panoramas [Agarwala et al. 2005]. Others have confirmed [Levin et al. 2004; Zomet et al. 2006; Goldman and Chen 2005] that gradient-domain compositing is a crucial component in state-of-the-art techniques for seamless panoramic stitching after the images have been aligned. Finally, Wang et al. [2004] were the first to adapt gradient-domain compositing to video.

The technique described in this paper should be applicable in all of the compositing systems just described. Gradient-domain techniques are not only used for compositing image regions, however. Various operations can be performed by creating an image that best matches a specified gradient-field; recent examples include high dynamic range (HDR) compression [Fattal et al. 2002], intrinsic image recovery [Weiss 2001], shadow removal [Finlayson and Drew 2002], flash artifact correction [Agrawal et al. 2005], reproducing photographic look [Bae et al. 2006], and alpha matting [Sun et al. 2004]. One drawback of our approach is that increases in efficiency will only occur if the problem can be transformed into a space where the solution is mostly smooth, and the pattern of smoothness can be predicted *a priori*. This transformation may not be possible for all problems, however, and the degree of smoothness will affect the gains in efficiency. We discuss the possibility of more widely applying our approach in Section 5. Finally, computing a large linear system is not the only way to solve Poisson equations; Fourier transforms can be used to directly calculate a solution [Simchony et al. 1990]. However, this approach requires $O(n \lg n)$ time, and more importantly, $O(n)$ memory.

2.1 Mathematical formulation

Gradient-domain compositing is performed for a single color channel of an image by re-ordering the image pixels into a vector x and solving for the x that best matches the desired horizontal and vertical gradients ∇I_x, ∇I_y. There are various possibilities for choosing these gradients. When compositing a region from image I_A into image I_B [Pérez et al. 2003; Jia et al. 2006], the gradients inside this region are ∇I_A and the colors at the boundaries are fixed from image I_B. When simultaneously compositing multiple regions from multiple images, the simplest approach [Agarwala et al. 2004] is to use the gradient of the source image between any two pixels inside of one region, and the average of the gradients of the two source images between any two pixels that straddle a boundary between two regions.

Each horizontal and vertical gradient specifies a single linear constraint on two variables, all of which can be expressed in matrix form as

$$Ax = b \tag{1}$$

where x is of length n (one element for each pixel), and A has at most two non-zero elements per row. This system of equations is overconstrained, and thus the solution x that minimizes $Ax - b$ in a least squares sense is the solution to the normal equations

$$A^T Ax = A^T b \tag{2}$$

where the sparse, banded square matrix $A^T A$ has at most five non-zero elements per row. Since this linear system is large it is typically solved using an iterative solver such as conjugate gradient [Shewchuk 1994] whose inner loop performs the sparse matrix-vector multiplication $A^T Ax$. This multiplication is equivalent to applying the Laplacian[2] to x, and thus can be applied procedurally without storing $A^T A$. The initial condition x_0 of the iterative solver is typically set to the image that would result from simply copying colors (rather than gradients) from the source images (e.g., first row of Figure 1).

2.2 Scalability issues

Consider using this approach to composite a panorama. A top-end digital SLR can produce 16 megapixel images, so a panorama from

[2] At the boundaries of the image the Laplacian kernel will depend on the choice of boundary conditions; the most common are Neumann and Dirichlet [Saad 2003], both of which are supported by our approach.

several such photographs can easily contain 50 megapixels (and often more). At a bare minimum, solving the n-element linear system in equation (2) for one color channel requires storing five floating point vectors of length n — one vector for x, one for $A^T b$, and three temporary vectors of storage during conjugate gradient iterations. Assuming four bytes per float, computing one channel of a 50 megapixel panorama would require one gigabyte of memory for these five vectors alone. The running time of this basic conjugate gradient solver would also be painfully long. Pyramidal approaches such as multigrid accelerate convergence but require even more memory; Szeliski [2006], for example, reports memory footprints that are roughly doubled.

As we show in the next section, this memory consumption is simply unnecessary; we can achieve visually equivalent results by solving a dramatically smaller linear system.

3 Our approach

The key to our approach is to solve the problem in a reduced space by assuming certain regions of the solution are smooth. The solution vector x itself is a natural image that will not typically be very smooth. Observe, however, that the initial residual $b - Ax_0$ will be zero for any pixel not adjacent to a seam, since the colors of that pixel and its neighbors were copied from one image and thus already satisfy the gradient constraints. If we substitute for x the sum $x = x_0 + x_\delta$, where x_δ is the difference between the initial condition and final solution, the normal equations become

$$A^T A x_\delta = A^T (b - A x_0). \tag{3}$$

Note that the right hand side of this equation will be zero for any pixel not adjacent to a seam. Regions of an image with a zero Laplacian will be very smooth. Thus, we can see that the offset x_δ to the initial condition x_0 will be very smooth away from the seams between image regions, even if the final image x is not smooth anywhere. An example of this pattern of smoothness for x_δ is shown in the third row of Figure 1.

Once this pattern of smoothness is realized, it becomes obvious that representing each pixel in a smooth area with one variable is wasteful; these areas can accurately be interpolated with fewer variables with larger regions of support. We can imagine varying the resolution of a solution vector adaptively; high resolution could be used near seams, and progressively lower resolutions in areas farther away from seams.

To accomplish this vision, we transform the full resolution problem in equation (1) into a reduced space

$$AS y = b \tag{4}$$

by substituting $x = Sy$, where y is a vector of dimension m such that $m \ll n$, and S is an $m \times n$ matrix that transforms from the reduced to the full space. Since the solution will not be smooth near seams, we wish any pixel adjacent to a seam to be represented with a single variable, just as in the full-resolution problem. Pixels in smoother areas may be interpolated as a weighted sum of several elements of y. The regions of support for interpolating x from elements of y should be larger further away from the seams. Given the interpolation $x = Sy$, the normal equations for the offset y_δ are

$$S^T A^T A S y_\delta = S^T A^T (b - A S y_0). \tag{5}$$

Once the matrix $S^T A^T AS$ on the left and vector $S^T A^T (b - ASy_0)$ on the right are pre-computed, the inner loop of an iterative solver for this linear system becomes an $m \times m$ rather than $n \times n$ sparse matrix-vector multiplication.

To define the matrix S we adaptively subdivide the problem domain using a quadtree [Samet 1990] that is maximally subdivided to pixel-sized nodes along the seams (a visualization of such a quadtree can be seen in Figure 1). We represent the quadtree with a pointer-based tree where each non-leaf node has four children that subdivide space into four quadrants. The root node corresponds to the smallest square that can entirely contain the image domain and whose width is a power of two. Variables in the reduced space (elements of y_δ) are placed at the corners between leaf nodes, as shown in Figure 2. Each leaf node stores the indices of the variables at its four corners. To ensure a gradual reduction in resolution away from the seams, we force the quadtree to be *restricted*; no two nodes that share an edge may differ in tree depth by more than one.

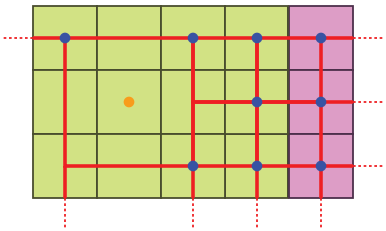

Figure 2 An inset of a grid of pixels in a larger gradient-domain composite. The green pixels are assigned to one source image, and the pink pixels to another; thus, a seam exists between them. The red lines are the boundaries of quadtree leaf nodes. The corners of quadtree nodes lie at pixel centers, and the quadtree is maximally subdivided along the seam. In the full linear system, each pixel is represented by one variable in the solution vector x. In the reduced linear system, variables in the solution vector y exist at the corners of the quadtree nodes (the blue dots), except along T-junctions where quadtree nodes of different sizes abut. We can interpolate the full solution x from a reduced solution y by computing $x = Sy$. This interpolation (which can be computed procedurally without ever building matrix S) is defined by the structure of the quadtree. Pixels that enclose a blue dot do not need to be interpolated; these values in x are just copied from the appropriate variable in y (which corresponds to a row in matrix S that is all 0 except for a single 1). Other pixels are interpolated; for example, the pixel enclosing the orange dot can be bi-linearly interpolated from the four corners of the enclosing quadtree node (which corresponds to a row of S with four non-zero values that sum to 1). One complication is that the lower-left corner of this node is not a variable in y, since it lies on a T-junction; the value here must first be linearly interpolated from the values of y at the blue dots above and below (not pictured).

Given this quadtree and the values of a vector y_δ, the interpolation $x_\delta = Sy_\delta$ can be computed with a single traversal of the quadtree, as shown in Figure 2. Note that the matrix S does not need to be explicitly built since it simply encodes a bi-linear interpolation from quadtree nodes to pixels. That is, each pixel of x_δ is set as the bi-linear interpolation of the values of y_δ at the four corners of the enclosing quadtree node. One exception is that we do not place variables at corners that lie at *T-junctions* between neighboring nodes of different depth; instead, T-junctions are interpolated from the two corners of the larger node along the horizontal stem of the "T".

3.1 Implementation details

Our overall algorithm can be be broken into three stages. In the input stage, the quadtree is constructed. Note that the colors of input image pixels appear only in the right-hand side of equation (5), and are non-zero only for pixels adjacent to a seam. Thus, in this input stage the colors of the relevant input images are stored only at leaf nodes bordering a seam. Each input image (or tile of an input image) can be immediately discarded after traversing the quadtree to store these colors.

The second stage is the computation of the linear system in equation (5). The right-hand side vector, which is computed only once per color channel, is non-zero only for variables that border seams.

Dataset	Mpixels	Vars (%)	Error RMS	Error Max	Time (s) QT	Time (s) HB	Time (s) LHB	Memory (MB) QT	Memory (MB) HB	Memory (MB) LHB
Plane	2.4	0.94	0.0108	0.36	3	371	26	13	96	227
St. Emilion	9.7	0.62	0.0132	1.37	9	3639	160	24	362	1044
Beynac	11.6	0.38	0.0103	1.25	8	3357	177	16	435	1252
Rainier	16.6	0.45	0.0157	1.13	14	6446	268	27	620	1790

Table 1 Performance of three algorithms for several gradient-domain compositing problems. For each dataset, we show the number of megapixels, the number of variables per color channel in the reduced linear system as a percentage of the total number of pixels, and the error between the solutions computed using the reduced and full linear systems (error is measured using the 8-bit red channel, with both an average per-pixel RMS error and the maximum error across all pixels). We show the time and memory performance of three algorithms: quadtree-based (QT), hierarchical basis preconditioning (HB), and locally adapted hierarchical basis preconditioning (LHB). Each panorama was stitched from five source images.

Since these variables always correspond to single pixels (Figure 2), the S and S^T matrices on the right-hand side do nothing but change the index of the variable. The right-hand side is therefore easy to compute procedurally in a single quadtree traversal. The matrix $S^T A^T A S$ on the left-hand side can also be computed procedurally in a single quadtree traversal by summing the contribution of the gradient constraint between each pair of neighboring pixels along the edges of quadtree nodes (the Laplacian of pixels internal to a quadtree node will be zero and thus can be ignored). We compute this matrix once and store it in a sparse, symmetric form. The four-neighbor quadtree traversal algorithm of Fuhrmann [1988] is useful during the setup of the linear system, which is then solved using preconditioned conjugate gradients. We precondition using a standard incomplete Cholesky factorization [Saad 2003], though even diagonal preconditioning is sufficient given the reduced size of the system. Note that there is never a need to allocate any $O(n)$ storage during this stage; the memory allocated is $O(m)$.

In the third and final output stage, the interpolated solution x_δ is added to the initial composite x_0. To avoid allocating the entire x_δ, the interpolation $x_\delta = S y_\delta$ can be performed independently for subregions (tiles) of the output image.

3.2 Scale of the reduced space

Since the input and output stages can be performed by streaming over arbitrarily small tiles of the source images, the memory and time requirements of our algorithm are bounded by the solution of an m-variable linear system. How small is m compared to n? The quadtree used here is equivalent to a *region quadtree* that represents a 2D array whose elements can be values in a small, discrete range. Dyer [1982] showed that the number of quadtree nodes in a region quadtree is $O(p)$, where p is the perimeter of the regions in the array, i.e., the total length of the seams. Our quadtree is restricted, but Moore [1995] showed that restriction only increases the number of nodes by a constant factor. Since m is linearly proportional to the number of leaf nodes in the quadtree, m is $O(p)$. The growth of p will depend on how the seams are chosen; for typical cases, we observe that p is $O(\sqrt{n})$. For example, if a single rectangular region is chosen from each one of a constant number of input images, p will be $O(\sqrt{n})$ since the perimeter of each region is upper-bounded by the perimeter of the composite (which is $O(\sqrt{n})$ assuming the width and height of the composite are related by a constant factor). The same upper-bound is true for a polygonal region with a constant number of sides. Seams chosen using graph cuts [Kwatra et al. 2003] will typically also be short since lengths are minimized by their cost functions, though one can imagine pathological inputs that would cause p to exceed $O(\sqrt{n})$.

Dataset	Mpixels	Vars (%)	# sources	Time (s)	Memory (MB)
Sedona	34.6	0.47	6	29	52
Edinburgh	39.7	1.15	25	122	123
Crag	62.7	0.47	7	78	96
RedRock	83.7	0.46	9	118	112

Table 2 Performance of quadtree-based gradient-domain compositing for several very large panoramas.

4 Experimental results

We compare the performance of our technique against our implementation of two other algorithms for several datasets of different sizes (Table 1), and show several results that were too large to compute in available memory using other algorithms (Table 2). Most of our results are panoramas whose seams were computed using hierarchical graph cuts [Agarwala et al. 2005], though the first result in Table 1 demonstrates image region copy-and paste with manually chosen seams. In the interest of space, most of our results can only be seen on the project web site, although the Rainier dataset is shown in Figure 1.

We compare against two approaches for solving the full linear system. The first is locally adapted hierarchical basis preconditioning (LHB) [Szeliski 2006], one of the fastest current approaches to solving gradient-domain problems. We also compare against the older hierarchical basis (HB) approach [Szeliski 1990], because unlike LHB, it requires no additional memory to perform preconditioning and thus can be considered a lower bound on the memory required to solve the full linear system. Along with performance comparisons, we also measure the error introduced by our reduction of the linear system by comparing its interpolated result (i.e., Sy) against the solution of the full linear system computed by LHB.

The reduced linear system is typically over 99% smaller. While it only approximates the full solution, the results are visually identical; when the images computed using both systems are rapidly flipped back and forth, no differences can be seen. Even when we scale the computed offsets by ten to generate the visualization in the third row of Figure 1, no differences are visible. The error values in Table 1 explain why. For color values that range from 0 to 255, the per-pixel RMS error is in the hundredths. The maximum error tells us that, once rounded to the nearest integer, color values differ by at most two for these examples. Differences this rare and small are simply not visible to the naked eye.

While the quality of the result remains the same, the reductions in both time and memory are dramatic. For all algorithms, the iterative solver was terminated when the sum of squared residuals (i.e., $||Ax-b||^2$ for the linear system $Ax = b$) was less than 10^{-11} times the number of pixels; this error tolerance is aggressively low, but gives us a high confidence that each result has converged. The LHB approach terminated in very few iterations (typically around 20 for our error tolerance irregardless of the number of pixels in the dataset), but the sheer size of the full system and the time required to setup the locally adapted pyramids causes its performance to be slower. Note that the performance numbers do not include resources consumed when reading and writing data from disk.

5 Future work

Our approach is very efficient at compositing image regions in the gradient-domain; an obvious extension is to perform gradient-domain compositing for video [Wang et al. 2004; Agarwala et al. 2005], where scalability concerns are even greater. This extension should be straightforward using octrees rather than quadtrees.

Our technique is effective because we can create an initial solution to the linear system whose residual is sparse. The same can be said

about several other gradient-domain problems, such as shadow removal [Finlayson and Drew 2002], removal of reflections in flash images [Agrawal et al. 2005], and reproduction of photographic look [Bae et al. 2006], since in these cases the desired gradient field largely matches the original image except for certain gradients that are damped or set to zero. This observation suggests that our approach could be used to improve their efficiency. However, it cannot be directly applied to other gradient-domain problems for which no such initial solution exists.

We also plan to explore an extension that may allow more efficient out-of-core reconstruction from general gradient fields. A solution could be computed for each tile of an image independently, thus creating an initial solution with non-zero residuals only along tile boundaries. Then, a quadtree could be subdivided along these tile boundaries and used to compute an offset to the initial solution that corrects the errors introduced by tile-by-tile computation.

6 Conclusion

While gradient-domain compositing is a remarkably effective technique for compositing image and video regions, it simply was not previously practical to use it for imagery of the large resolutions common even in consumer-level digital imaging. We have shown an approximate approach to gradient-domain compositing that yields visually identical results, yet can be computed in surprisingly little time and memory, even for very large composites. This efficiency improvement allowed us to use gradient-domain compositing in the new "Auto-Blend Layers" feature in Adobe® Photoshop® CS3. We hope that our technique will be one of many to address the scalability of algorithms for computational photography and video.

Acknowledgements: Thanks to Dan Goldman for fruitful discussions, Rick Szeliski for advice in implementing his preconditioner, and Dan Goldman, Michael Cohen, and David Salesin for help with the manuscript. Thanks to Jeff Chien for helping me transfer this research into Photoshop, and to Tobias Oberlies and Brian Curless for images.

References

AGARWALA, A., DONTCHEVA, M., AGRAWALA, M., DRUCKER, S., COLBURN, A., CURLESS, B., SALESIN, D., AND COHEN, M. 2004. Interactive digital photomontage. *ACM Transactions on Graphics 23*, 3 (Aug.), 294–302.

AGARWALA, A., ZHENG, K. C., PAL, C., AGRAWALA, M., COHEN, M., CURLESS, B., SALESIN, D. H., AND SZELISKI, R. 2005. Panoramic video textures. *ACM Transactions on Graphics 24*, 3 (Aug.), 821–827.

AGARWALA, A., AGRAWALA, M., COHEN, M., SALESIN, D., AND SZELISKI, R. 2006. Photographing long scenes with multi-viewpoint panoramas. *ACM Transactions on Graphics 25*, 3 (July), 853–861.

AGRAWAL, A., RASKAR, R., NAYAR, S. K., AND LI, Y. 2005. Removing photography artifacts using gradient projection and flash-exposure sampling. *ACM Transactions on Graphics 24*, 3 (Aug.), 828–835.

BAE, S., PARIS, S., AND DURAND, F. 2006. Two-scale tone management for photographic look. *ACM Transactions on Graphics 25*, 3 (July), 637–645.

BOLZ, J., FARMER, I., GRINSPUN, E., AND SCHRÖDER, P. 2003. Sparse matrix solvers on the GPU: Conjugate gradients and multigrid. *ACM Transactions on Graphics 22*, 3 (July), 917–924.

DYER, C. 1982. The space efficiency of quadtrees. *Computer Graphics and Image Processing 19*, 4 (Aug.), 335–348.

FATTAL, R., LISCHINSKI, D., AND WERMAN, M. 2002. Gradient domain high dynamic range compression. *ACM Transactions on Graphics 21*, 3, 249–256.

FINLAYSON, G., AND DREW, S. H. M. 2002. Removing shadows from images. In *European Conference on Computer Vision (ECCV 02)*, 823–831.

FUHRMANN, D. R. 1988. Quadtree traversal algorithms for pointer-based and depth-first representations. *IEEE Transactions on Pattern Analysis and Machine Intelligence 10*, 6, 955–960.

GEORGIEV, T. 2004. Photoshop healing brush: a tool for seamless cloning. In *Workshop on Applications of Computer Vision (ECCV 2004)*, 1–8.

GOLDMAN, D. B., AND CHEN, J.-H. 2005. Vignette and exposure calibration and compensation. In *International Conference on Computer Vision (ICCV 05)*, 899–906.

HAYS, J., AND EFROS, A. 2007. Scene completion using millions of photographs. *ACM Transactions on Graphics 26*, 3, To appear.

JIA, J., SUN, J., TANG, C.-K., AND SHUM, H.-Y. 2006. Drag-and-drop pasting. *ACM Transactions on Graphics 25*, 3 (July), 631–637.

KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics 22*, 3, 277–286.

LAND, E. H. 1977. The retinex theory of color vision. *Scientific American 237*, 6, 108–128.

LEVIN, A., ZOMET, A., PELEG, S., AND WEISS, Y. 2004. Seamless image stitching in the gradient domain. In *European Conference on Computer Vision (ECCV 04)*, 377–389.

LOSASSO, F., GIBOU, F., AND FEDKIW, R. 2004. Simulating water and smoke with an octree data structure. *ACM Transactions on Graphics 23*, 3 (Aug.), 457–462.

MOORE, D. 1995. The cost of balancing generalized quadtrees. In *Proceedings of the Third ACM Symposium on Solid Modeling and Applications*, 305–312.

PALMER, S. E. 1999. *Vision Science: Photons to Phenomenology*. The MIT Press.

PÉREZ, P., GANGNET, M., AND BLAKE, A. 2003. Poisson image editing. *ACM Transactions on Graphics 22*, 3 (July), 313–318.

SAAD, Y. 2003. *Iterative methods for sparse linear systems*, 2nd ed. Society for Industrial and Applied Mathematics (SIAM).

SAMET, H. 1990. *Applications for spatial data structures: computer graphics, image processing, and GIS*. Addison-Wesley.

SHEWCHUK, J. R. 1994. An introduction to the conjugate gradient method without the agonizing pain. Tech. Rep. CS-94-125, Carnegie Mellon University.

SIMCHONY, T., CHELLAPPA, R., AND SHAO, M. 1990. Direct analytical methods for solving Poisson equations in computer vision problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence 12*, 5, 435–446.

SUN, J., JIA, J., TANG, C.-K., AND SHUM, H.-Y. 2004. Poisson matting. *ACM Transactions on Graphics 23*, 3 (Aug.), 315–321.

SZELISKI, R., AND SHUM, H.-Y. 1996. Motion estimation with quadtree splines. *IEEE Transactions on Pattern Analysis and Machine Intelligence 18*, 12, 1199–1210.

SZELISKI, R. 1990. Fast surface interpolation using hierarchical basis functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence 12*, 6, 513–528.

SZELISKI, R. 2006. Locally adapted hierarchical basis preconditioning. *ACM Transactions on Graphics 25*, 3 (July), 1135–1143.

TOLEDO, S. 1999. A survey of out-of-core algorithms in numerical linear algebra. In *External Memory Algorithms*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science. 161–180.

WANG, H., RASKAR, R., AND AHUJA, N. 2004. Seamless video editing. In *Proceedings of the International Conference on Pattern Recognition (ICPR)*, 858–861.

WEISS, Y. 2001. Deriving intrinsic images from image sequences. In *International Conference On Computer Vision (ICCV 01)*, 68–75.

ZOMET, A., LEVIN, A., , PELEG, S., , AND WEISS, Y. 2006. Seamless image stitching by minimizing false edges. *IEEE Transactions on Image Processing 15*, 4, 969–977.