Fast Poisson Blending using Multi-Splines

Richard Szeliski, Matthew Uyttendaele, and Drew Steedly
Microsoft Research and Microsoft

Abstract

We present a technique for fast Poisson blending and gradient domain compositing. Instead of using a single piecewise-smooth offset map to perform the blending, we associate a separate map with each input source image. Each individual offset map is itself smoothly varying and can therefore be represented using a low-dimensional spline. The resulting linear system is much smaller than either the original Poisson system or the quadtree spline approximation of a single (unified) offset map. We demonstrate the speed and memory improvements available with our system and apply it to large panoramas. We also show how robustly modeling the multiplicative gain rather than the offset between overlapping images leads to improved results, and how adding a small amount of Laplacian pyramid blending improves the results in areas of inconsistent texture.

1. Introduction

While Poisson blending [24] (also known as *gradient domain compositing*) was originally developed to support the seamless insertion of objects from one image into another [24, 17], it has found widespread use in hiding seams due to exposure differences in image stitching applications [22, 3, 2, 16]. Poisson blending usually produces good results for these applications. Unfortunately, the resulting two-dimensional optimization problems require large amounts of memory and time to solve [18].

A number of approaches have been proposed in the past to speed up the solution of the resulting sparse system of equations. One approach is to use multigrid [21, 18] or multi-level preconditioners [27] (which can be implemented on GPUs [6, 8, 23]) to reduce the number of iterations required to solve the system. Agarwala [2] proposed using a quadtree representation of the offset field, which is the difference between the original unblended images and the final blended result. Farbman *et al.* [12] use mean value coordinates (MVC) defined over an adaptive triangulation of the cloned region to interpolate the offset field values at the region boundary. It is also possible to solve the Poisson blending problem at a lower resolution, and to then upsample the resolution while taking the location of seams into account [19]. Finally, Fourier techniques can be used to solve Poisson problems [4], but these require careful treatment of image boundary conditions and are still log-linear in the number of pixels.

In this paper, we propose an alternative approach, which further reduces the number of variables involved in the system. Instead of using a single offset field as in [2, 12], we associate a separate low-resolution offset field with each source image. We then simultaneously optimize over all of the (coupled) offset field parameters. Because each of the offset fields is represented using a low-dimensional spline, we call the resulting representation a *multi-spline*.

The basis of our approach is the observation that the offset field between the original unblended solution and the final blended result is piecewise smooth except at the seams between source regions [24, 2]. In his paper on efficient gradient-domain compositing, Agarwala [2] exploits this property to represent the offset field using a quadtree. In this paper, we observe that if the offset field is partitioned into separate per-source correction fields, each of these will be truly smooth rather than just piecewise smooth.

Our method is thus related to previous work in exposure and vignetting compensation [11, 15, 19], as it computes a per-image correction that reduces visible seams at region boundaries. Motivated by this observation, we investigate the use of multiplicative rather than additive corrections and show that these generally produce better results for image stitching applications. We also show how adding a small amount of Laplacian pyramid blending [9] can help mask visual artifacts in regions where inhomogeneous textures are being blended, e.g., water waves that vary from shot to shot.

The remainder of this paper is structured as follows. First, we formulate the Poisson blending problem and show how it can be reformulated as the computation of a piecewise-smooth offset field (Section 2). Next, we introduce the concept of multiple offset maps (Section 3) and show how these can be represented using tensor product splines (Section 4). In Section 5, we discuss efficient methods for solving the resulting sparse set of linear equations.[1] In Section 6, we apply our technique to a variety of large-scale image stitching problems, demonstrating both the speed and memory improvements available with our technique, as well as the quality improvements available from using multiplicative (gain) compensation. We close with a discussion of the results and ideas for possible extensions.

2. Problem formulation

The Poisson blending problem can be written in discrete form as

$$E_1 = \sum_{i,j} s^x_{i,j}[f_{i+1,j}-f_{i,j}-g^x_{i,j}]^2+s^y_{i,j}[f_{i,j+1}-f_{i,j}-g^y_{i,j}]^2, \tag{1}$$

where $f_{i,j}$ is the desired Poisson blended (result) image, $g^x_{i,j}$ and $g^y_{i,j}$ are the target gradient values, and $s^x_{i,j}$ and $s^y_{i,j}$ are the (potentially per-pixel) gradient constraint (smoothness) weights. (This notation is adapted from [27].)

In the original formulation [24], the weights are all set uniformly, and the gradients are computed from the gradients of the source image being blended in, with additional hard constraints along the boundary of the cut-out region to match the enclosing image. In the general multi-image formulation of Poisson blending [3], the gradients are obtained from the gradients of whichever image is being composited inside a given region,

$$\begin{aligned} g^x_{i,j} &= u^{l_{i,j}}_{i+1,j} - u^{l_{i,j}}_{i,j}, && (2)\\ g^y_{i,j} &= u^{l_{i,j}}_{i,j+1} - u^{l_{i,j}}_{i,j}, && (3) \end{aligned}$$

where $\{u^1 \dots u^L\}$ are the original *unblended* (source) images and $l_{i,j}$ is the *label* (indicator variable) for each pixel, which indicates which image is being composited. At the boundaries between regions, the average of the gradients from the two adjacent images is used,

$$\begin{aligned} g^x_{i,j} &= (u^{l_{i,j}}_{i+1,j} - u^{l_{i,j}}_{i,j} + u^{l_{i+1,j}}_{i+1,j} - u^{l_{i+1,j}}_{i,j})/2, && (4)\\ g^y_{i,j} &= (u^{l_{i,j}}_{i,j+1} - u^{l_{i,j}}_{i,j} + u^{l_{i,j+1}}_{i,j+1} - u^{l_{i,j+1}}_{i,j})/2. && (5) \end{aligned}$$

Note how these equations reduce to the previous case (2) and (3) on the interior, since the indicator variables are the same. We then substitute (2–5) into (1) and minimize the resulting cost function. The resulting function f reproduces the high-frequency variations in the input images while feathering away low-frequency intensity offsets at the seam boundaries (Figure 1a).

The per-pixel weights can be tweaked to allow the final image to match the original image with less fidelity around strong edges [3], where the eye is less sensitive to variations, resulting in what is sometimes called the *weak membrane* [27]. (This can also be used to perform dynamic range compression [14], which can be advantageous when compositing images with widely different exposures.) In this paper, as in [3, 2, 5], we set the weights to be constant inside each source region, but allow them to be weaker along boundary pixels where the two source gradients disagree [5]. We first compute the difference between corresponding image gradients

$$\Delta g^x_{i,j} = (u^{l_{i,j}}_{i+1,j} - u^{l_{i,j}}_{i,j}) - (u^{l_{i+1,j}}_{i+1,j} - u^{l_{i+1,j}}_{i,j}) \tag{6}$$

and then set the horizontal smoothness weight to

$$s^x_{i,j} = \frac{1}{(1+a|\Delta g^x_{i,j}|)^b}, \tag{7}$$

with corresponding equations for the y gradients. The parameter $a = 16$ is used to scale the color values into an appropriate range, while the $b = 9$ parameter controls the strength of the weight.

If only gradient constraints are used (1), the gradient-domain reconstruction problem is underconstrained. Pérez *et al.* [24] use hard constraints along the region boundary, while Agarwala *et al.* [3] have the user specify a single pixel value to match. In our work, we add a weak constraint towards the colors in the original image $u^{l_{i,j}}_{i,j}$,

$$E_0 = \sum_{i,j} w_{i,j}[f_{i,j} - u^{l_{i,j}}_{i,j}]^2, \tag{8}$$

typically with $w_{i,j} = 10^{-7}$, which reduces unwanted low-frequency variations in the result and helps ensure that the final composite does not get too light or dark.

2.1. Offset formulation

As noted in [24, Eqn.(5)] and [2, Eqn.(3)], we can replace the solution $\{f_{i,j}\}$ with an *offset* from the original (unblended) image,

$$f_{i,j} = u^{l_{i,j}}_{i,j} + h_{i,j} \tag{9}$$

and solve for the offset image $\{h_{i,j}\}$ instead. The new criterion being minimized becomes

$$\begin{aligned} E_2 = \sum_{i,j} \; & s^x_{i,j}[h_{i+1,j} - h_{i,j} - \tilde{g}^x_{i,j}]^2 + && (10)\\ & s^y_{i,j}[h_{i,j+1} - h_{i,j} - \tilde{g}^y_{i,j}]^2 + w_{i,j}[h_{i,j}]^2, \end{aligned}$$

where the modified gradients $\tilde{g}^x_{i,j}$ and $\tilde{g}^y_{i,j}$ are zero away from the boundaries and

$$\begin{aligned} \tilde{g}^x_{i,j} &= (u^{l_{i,j}}_{i,j} - u^{l_{i+1,j}}_{i,j} + u^{l_{i,j}}_{i+1,j} - u^{l_{i+1,j}}_{i+1,j})/2, && (11)\\ \tilde{g}^y_{i,j} &= (u^{l_{i,j}}_{i,j} - u^{l_{i,j+1}}_{i,j} + u^{l_{i,j}}_{i,j+1} - u^{l_{i,j+1}}_{i,j+1})/2, && (12) \end{aligned}$$

[1] An earlier version of this paper [29] has a more in-depth discussion of various alternative sparse solvers.

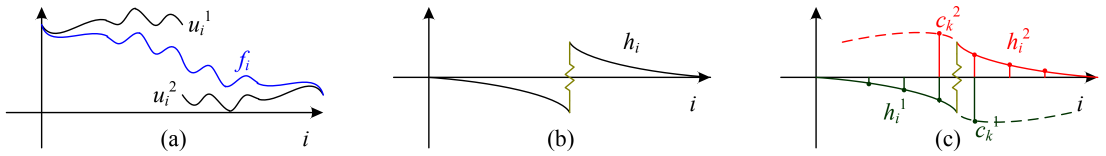

Figure 1. One dimensional examples of Poisson blending and offset maps: (a) the original Poisson blend of two source images u_i^1 and u_i^2 produces the blended function f_i; (b) the offset image h_i is fitted to zero gradients everywhere except at the source image discontinuity, where it jumps by an amount equal to the average difference across the region boundary; (c) the multiple offset images h_i^1 and h_i^2, each of which is smooth, along with the inter-image constraint at the boundary; the offsets are defined by the spline control vertices c_k^1 and c_k^2.

at the boundaries between regions.

This new problem has a natural interpretation: the offset value should be everywhere smooth, except at the region boundaries, where it should jump by an amount equal to the (negative) *average difference* in intensity between the overlapping source images. The resulting offset function is piecewise smooth (Figure 1b), which makes it amenable to being represented by a quadtree spline, with smaller grid cells closer to the region boundaries [2] or with mean value coordinate interpolation [12].

3. Multiple offset maps

In this paper, instead of using a single offset map, as suggested in [24, 2, 12], we use a different offset map for each source image, i.e.,

$$f_{i,j} = u_{i,j}^{l_{i,j}} + h_{i,j}^{l_{i,j}}, \tag{13}$$

where the $\{h^1 \dots h^l\}$ are now the *per-source image offset maps* (see Figure 1c).

The optimization problem (10) now becomes

$$E_3 = \sum_{i,j} s_{i,j}^x [h_{i+1,j}^{l_{i+1,j}} - h_{i,j}^{l_{i,j}} - \tilde{g}_{i,j}^x]^2 + \tag{14}$$
$$s_{i,j}^y [h_{i,j+1}^{l_{i,j+1}} - h_{i,j}^{l_{i,j}} - \tilde{g}_{i,j}^y]^2 + w_{i,j} [h_{i,j}^{l_{i,j}}]^2,$$

Notice that in this problem, whenever two adjacent pixels, say (i,j) and $(i+1,j)$ come from the same source and hence share the same offset map, the gradient $\tilde{g}_{i,j}^x$ is 0, and so the function is encouraged to be smooth. When two adjacent pixels come from different regions, the *difference* between their offset values is constrained to be the average difference in source values at the two pixels (11). This is illustrated schematically in Figure 1c.

What is the advantage of re-formulating the problem using a larger number of unknowns? There is none if we keep all of the $h_{i,j}^l$ as independent variables.

However, under normal circumstances, e.g., when working with log intensities and multiplicative exposure differences, each of the individual per-source offset maps will be *smooth*, and not just *piecewise smooth* as in the case of a single offset map. Therefore, each offset map can be represented at a much lower resolution, as we describe next.

4. Spline offset maps

To take advantage of the smoothness of each offset image, we represent each map with a tensor-product spline that covers the visible extent of each region, as shown in Figure 2. The choice of pixel spacing (subsampling) S is problem dependent, i.e., it depends on the amount of unmodeled variations in the scene and acquisition process, e.g., the severity of lens vignetting or the amount of inconsistent texture along the seam, but is largely independent of the actual pixel (sensor) resolution. We can either align each grid with each region's bounding box (Figure 2a) or use a globally consistent alignment (Figure 2b). We use the latter, since it makes the nested dissection algorithm discussed in Section 5 easier to implement.

Once we have chosen S and the control grid locations, we can re-write each pixel in an individual offset map as a linear combination of the per-level spline control vertices $c_{k,m}^l$ (Figure 1c),

$$h_{i,j}^l = \sum_{km} c_{k,m}^l B(i - kS, j - mS) \tag{15}$$

where

$$B(i,j) = b(i)b(j) \tag{16}$$

is a 2D tensor product spline basis and $b(i)$ is a 1-D B-spline basis function [13]. For example, when bilinear (first order) interpolation is used, as is the case in our experiments, the first order 1-D B-spline is the usual tent function, and each pixel is a linear blend of its 4 adjacent control vertices (Figure 2c).

The values of $h_{i,j}^l$ in (15) can be substituted into (14) to obtain a new energy function (omitted for brevity) that only depends on the spline control variables $c_{k,m}^l$. This new energy function can be minimized as a sparse least squares system to compute a smooth spline-based approximation to the offset fields. Once the sparse least squares system has been solved, as described in Section 5, the per-pixel offset values can be computed using regular spline interpolation (15).

The actual inner loop of the least squares system setup simply involves iterating over all the pixels, pulling out the

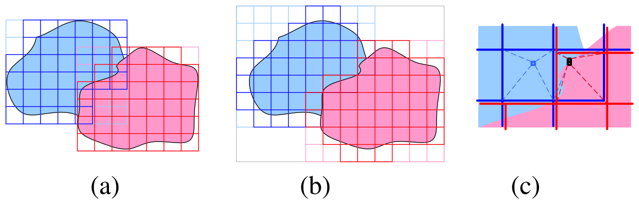
Figure 2. A spline grid is overlaid on top of each source region: (a) the grids are aligned with each region's bounding box; (b) the grids are aligned with the final composite bounding box (shown in gray). The grid cell and control variables (nodes at cell corners) that are inactive are shown in pale colors. (c) Detail inside two of the spline patches: inside the left square (blue), each pixel depends on four neighboring control vertices; along a seam boundary (mixed colors), each pixel depends on eight.

$(K+1)^d$ non-zero B-spline basis function values (where K is the order of the interpolant, and d is the dimensionality of the field), forming each of the linear equations in the control variables inside each squared term, and then updating the appropriate entries in the normal equations (stiffness matrix and right-hand side) [28, 27].

Figure 2c illustrates this process for bilinear splines, where each offset value $h^l_{i,j}$ (blue pixel inside the large blue square) depends on its four neighboring control vertices. Thus, each gradient constraint in (14) depends on either four or eight control vertices, the latter occurring only at seam boundaries where offsets from two different spline maps are being compared (i.e., the pair of black pixels inside the mixed color region). Note that spline vertices that do not influence any visible pixels can be eliminated from the variable set and are shown as pale colors in Figures 2a-b.

4.1. Simplifying the constraints

Inside spline patches where all the pixels come from the same source (the blue patch in Figure 2c) and the smoothness and data weights are homogeneous, $s^x_{i,j} = s^y_{i,j} = s$ and $w_{i,j} = w$, we can pre-compute the effect of all the individual per-pixel gradient and smoothness constraints ahead of time. This is similar to the process of analytic integration performed during finite element analysis to determine the effects of continuous deformations on the discrete control variables [30]. Performing this analysis for a bilinear interpolant yields a *nine-point stencil*, i.e., a system of equations where each variable is coupled not only to its horizontally and vertically adjacent neighbors but also to vertices that are diagonally adjacent.

In our current implementation, we instead adopt a simpler approach and assume, for the purpose of computing the internal smoothness constraints, that the interpolating spline uses conforming triangular linear elements, as in [30, 27]; for the data constraint, we use a piecewise constant interpolant. This results in a simple per-offset field energy function that is a coarsened version of the original fine-level Poisson blending energy,

$$E^l = \sum_{k,m} s[c^l_{k+1,m} - c^l_{k,m}]^2 + s[c^l_{k,m+1} - c^l_{k,m}]^2 + S^2 w [c^l_{k,m}]^2. \tag{17}$$

To further speed up the formulation of the least squares system, we apply this same discrete energy to *all* spline patches within each offset layer. We then add in the original gradient constraints (14) only along seam boundary pixels, i.e., pairs of pixels that have two different labels, as shown by the pair of black pixels in Figure 2c.

4.2. Multiplicative (gain) offsets

The observant reader may have noticed that multi-spline offset fields are just a generalization of the bias-gain correction fields commonly used to adjust the exposures of images before blending [11, 15, 19]. A single linear spline patch per source image is equivalent to the commonly used affine intensity variation model. If full spline fields are used, one can view our new technique as simply constraining the spline correction fields to map overlapping pixels near boundary regions to the same value.

The important difference, however, between our derivation from the original Poisson blending equations and a more heuristic implementation is that our approach directly tells us how to set the tradeoff between the smoothness in the final correction field and the strength of the seam matching term. Without this balance, the resulting correction fields can result in composites with either visible tears or flat spots.

Viewing Poisson blending as the computation of per-image correction fields suggests that computing multiplicative gain fields may be preferable to computing additive offset fields [31, 18]. In fact, most visible seams in panoramas are due to camera exposure variation, vignetting, or illumination changes (the sun going behind a cloud), all of which are better modeled as multiplicative gains rather than additive offsets. This is true even if the images are gamma-corrected, since the resulting images are still related by a multiplicative factor, i.e., $I_2 = kI_1 \Rightarrow I_2^\gamma = k^\gamma I_1^\gamma$.

To estimate a multiplicative gain, we simply take the logarithm of each input image before computing the seam difference, or, equivalently, take the logarithm of the ratio of overlapping seam pixels. (Dark pixels can be clamped to a minimum value such as 1.) The resulting computed offset field is then exponentiated and used as a multiplicative gain field.

In our experience, multiplicative gain correction works very well when the inputs are in the camera sensor (RAW) domain. However, JPEG images produced by cameras and commercial software have usually undergone black box processing [1] that causes them to no longer fit a purely ad-

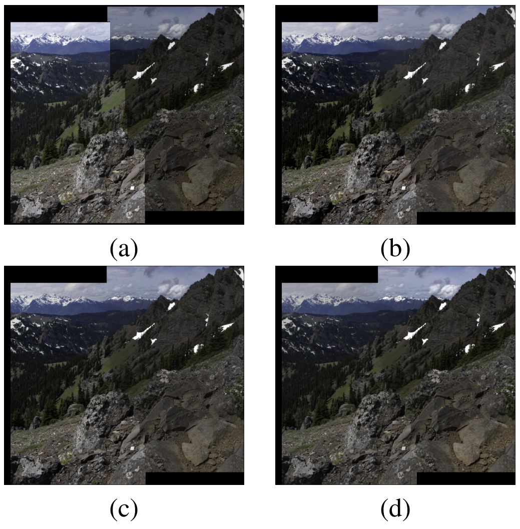
Figure 3. Comparison of multiplicative gain vs. additive offset blending: (a) unblended image; (b) additive offset blending; (c) square root offset blending; (d) multiplicative gain blending. Note how the additive result has a visible contrast change across the seam.

ditive or multiplicative model. For this reason we prefer to compute the offset field in a log/linear domain. In our experience, the square root function approximates the log function well for bright values and approaches linear for dark values. In our implementation, we apply the square root function (with pixels normalized to a $[0, 1]$ range) to each pixel before computing the seam differences. The resulting offset field is then added to the square root pixel values and the result is then squared.

Figure 3 compares the result using the multiplicative gain approach vs. the traditional additive Poisson blending approach. Because the additive offset does not model the differing amounts of contrast in the two source images (which are related by a multiplicative exposure difference), the blended result in Figure 3b has a visible contrast change in the vicinity of the seam (more muddy looking colors to the right of the seam, which are better seen in the supplementary materials).

4.3. Laplacian pyramid blending

While Poisson blending does a good job of compensating for slowly varying differences between images such as exposure or lighting changes, it has a harder time hiding artifacts due to inhomogeneous (inconsistent) textures in the two images. Consider, for example, the wave-tossed waters shown in Figure 4. While Poisson blending does a good job of disguising low-frequency color and intensity differences, the differences between the individual wave patterns result in visible seams. Applying multi-band Laplacian pyramid blending [9, 7] with a small number of pyramid levels to

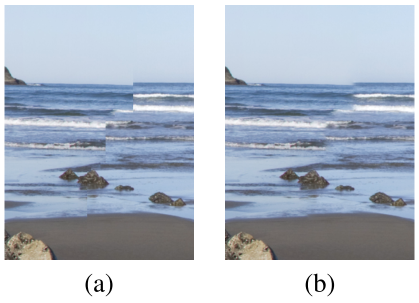
Figure 4. Using Laplacian pyramid blending to disguise texture differences: (a) image robustly blended in the square root pixel domain; (b) with the addition of three-level Laplacian pyramid blending. Note how the differences in the wave patterns and small misalignments are effectively masked.

the gain or bias-compensated images helps disguise these differences. Since our multi-spline Poisson blending effectively handles the low frequency differences between the images, there is no need to use a large number of pyramid levels. The multi-level pyramid filtering operations can therefore be restricted to a narrow band around the seams, which can lead to large computational savings.

4.4. Blending Gigapixel images

To make our technique even more efficient, instead of computing the seam costs at the final image resolution, we compute these costs at the same resolution as the graph cut optimization performed in [20], which is $^1/_8$th the horizontal and vertical resolution of the final panorama.

The decision to accumulate the seam costs on a lower resolution image is actually well-justified. Since the relative contribution of each seam constraint to the spline vertices is a slowly varying function, summing these contributions over a slightly coarser grid than the pixels (but still finer than the spline) does not affect the results very much. Because we are computing a least squares correction, summing up the least squares contributions over regions does not affect the final cost, except for the replacement of the spline weights with a slightly more discretized version.

Another way of looking at this is that if we are estimating the offset or gain adjustment between overlapping images, a similar result will be obtained if we look at lower-resolution versions of these images, so long as the lowered resolution is still significantly higher than the spline grid.

Once we have computed our multi-spline correction fields [29], these are then upsampled to the final Gigapixel resolution during the tile-based final compositing process [20].

5. Solving the system

A variety of techniques can be used to solve the small sparse positive definite system of equations arising from the multi-spline correction fields. For large sparse system, iterative techniques (potentially enhanced with multi-grid or multi-level preconditioning) can be used [25]. When the systems are smaller, direct factorization techniques such as Cholesky decomposition are more efficient [10].

Because his sparse systems are larger, Agarwala [2] uses the conjugate gradient sparse iterative solver, which partially accounts for his longer run times. Because our multi-spline systems are much smaller, we use direct techniques. The efficiency of these techniques depends on the amount of *fill-in* during the factorization process, which can be reduced by an appropriate reordering of the variables [10, 29].

For two-dimensional grid problems, *nested dissection*, which recursively splits the problem along small length rows or columns, results in good performance, i.e., $O(n \log n)$ space and $O(n^{3/2})$ time (or better for asymmetrically shaped domains), where n is the number of variables. In order for this technique to work, we need to ensure that all the spline variables line up in the same rows and columns, which is why we use the aligned spline grid shown in Figure 2b.

6. Experiments

In order to validate our approach and to see how much of a speedup could be expected, we first obtained the four large panoramas shown in Figure 6 from the author of [2]. For these images, we used an additive offset field to match the results presented in [2] as closely as possible. We also used a spline spacing of $S = 64$ and bilinear splines.

The results of running our algorithm (in January, 2008) [29] on these four data sets are shown in Table 2. Our multi-spline technique is about 5–10× faster and requires about 10× less memory than the quadtree-based approach developed in [2]. The two techniques produce results of comparable visual quality, as can be seen by inspecting the large full-size images provided in the supplementary material.

Grid size S	RMS error	max error	solve time (s)
8	0.0886	11.20	163.383
16	0.1039	12.80	11.714
32	0.1841	13.70	0.851
64	0.2990	14.40	0.070
128	0.4118	13.90	0.019

Table 1. RMS and maximum error comparisons to the ground truth Poisson blend for different grid sizes S, along with the linear system solution time (in seconds); the time for setup and rendering the final offset fields is about 2 seconds. For these experiments, we used the 9.7 Mpixel *St. Emilion* dataset shown in Figure 5.

Table 1 shows how the RMS (root mean square) and maximum error (in gray levels) in the solution depend on the grid size S. For these experiments, we used the solution to the *St. Emilion* data set provided by Agarwala [2] (Figure 5) as our ground truth. We then ran our fast multi-spline-based solver using a variety of grid sizes, $S = \{8, 16, \ldots, 128\}$ and computed both the RMS and maximum error between the offset field we computed and the full solution. As you can see, the RMS error grows steadily with the grid size, while the maximum error does not vary that much. Visual inspection of the full-resolution results (which are available as part of the supplementary materials) shows that the maximum error is concentrated at isolated pixels along seam boundaries where highly textured regions are mis-aligned. Fortunately, these "errors" are masked by the textures themselves, so that the final blended images appear of identical quality to the eye.

Next, we applied our technique to the Seattle Skyline image shown in Figure 4 of [20], using the multiplicative gain (log intensity) formulation because of the large exposure differences. In this case, because the seam costs were computed on a $^1/_8$th (on side) resolution image, the seam cost evaluation (shown as Setup in Table 2) and system solving times as well as the memory requirement are comparable to those of the 84 Mpixel RedRock panorama. The rendering time required to read back and warp the source images, apply the spline-based correction, and write out the resulting tiles is significantly longer.

A cropped portion of our result is shown in Figure 5, and the unblended, offset, and blended images at the $^1/_8$th working resolution, along with some cropped portions of the final Gigapixel image are shown in the supplementary materials.

The most visible artifacts in these results, besides the saturated regions and gross misalignments caused by the moving crane, are the occasional seams visible in the sky regions near dark buildings, which are due to some of the original source images having saturated pixels (usually in the blue channel) in these regions. Unfortunately, since the values at these pixels do not reflect the true irradiance, the multiplicative gain computed in areas that border unsaturated pixels is inconsistent, and cannot simultaneously hide both kinds of seams.

7. Discussion and extensions

As we can see from our experiments, the biggest difference between our multi-spline approach and full Poisson blending (and its quadtree approximation) is that we enforce piecewise smoothness in both the x and y dimensions, whereas Poisson blending can tolerate irregular offsets *along* the seam. While our approach can occasionally lead to artifacts, e.g., in images that are not log-linear, Poisson blending can introduce different artifacts, such as

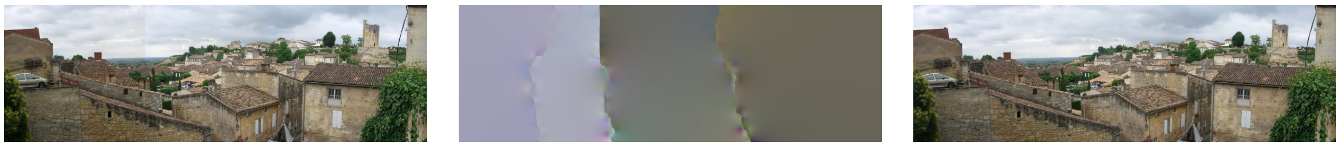
(a) (b) (c)

Figure 5. Fast Poisson blending using multi-splines: (a) unblended composite; (b) piecewise smooth multi-spline offset image; (c) final blended composite.

			Quadtree			Multi-spline			Setup		Solve		Render
Dataset	#	Mpix	V (%)	T (s)	M	V (%)	T (s)	M	T (s)	M	T (s)	M	T (s)
Sedona	6	34.6	0.47	29	52	0.0271	7	4	3.33	3	0.28	5	2.70
Edinburgh	25	39.7	1.15	122	123	0.0315	9	10	3.99	10	0.41	7	3.84
Crag	7	62.7	0.47	78	96	0.0271	12	7	6.16	6	0.54	9	4.82
RedRock	9	83.7	0.46	118	112	0.0270	16	10	8.11	8	0.75	13	6.42
Seattle	650	3186.9				0.0009	8+	57	6.35	57	1.42	34	56m

Table 2. Performance of the quadtree vs. multi-spline based solver. The first three columns list the dataset name, the number of source images, and the number of pixels. The next three columns show the results for the quadtree-based acceleration, including the ratio of variables to original pixels (as a percentage), the total run time (in seconds), and the memory usage (in Mbytes). The next three columns show the corresponding results for our multi-spline based approach (for all of our experiments, $S = 64$). Our results are roughly 5–10$\times$ faster and smaller. The final sets of columns break down the time and memory requirements of the three multi-spline blending stages, namely the setup (computation of seam boundary constraints), the direct solution using nested dissection, and the final rendering (compensation) stage. The numbers in the Quadtree column are from [2], which does not report the processor used. The numbers in the other columns are from experiments run single-threaded on a 2.40 Intel GHz Core$^{\text{TM}}$2 Duo processor with 4GB RAM purchased in March, 2007, and therefore likely comparable to the processor used by Agarwala. Note that the Gigapixel Seattle total time (8+) does not include the i/o bound rendering stage, which took 56 minutes to produce the final image tile set.

"ruffles" that sometime propagate away from seam boundaries when they disagree. Improving the alignment between images using an optic-flow deghosting technique [26] followed by robustly estimating the mapping between overlapping images is likely to futher improve the results.

In terms of computational complexity, as the resolution of photographs continues to increase, our multi-spline based approach has better scaling properties that the quadtree based approach. Because the number of spline control vertices depends on the smoothness of the unmodeled inter-exposure variation and not the pixel density, we expect it to remain fixed. In the quadtree-based approach, the number of variables increases linearly with the on-side (as opposed to pixel count) resolution. In order to further speed up the linear system solving part of our algorithms, we are also investigating hierarchically preconditioned conjugate gradient descent [27].

8. Conclusions

In this paper, we have developed a new approach to gradient domain compositing that allocates a separate smoothly varying spline correction field for each source image. We also investigated the benefits of using a multiplicative gain formulation over the more traditional additive offset formulation. Using our approach, we obtain linear systems an order of magnitude smaller than those obtained with a quadtree representation of a single offset map, while producing results of comparable visual quality. We also suggest areas for further investigations into better quality algorithms for seam blending.

References

[1] Adobe. Digital Negative (DNG) Specification. Version 1.3.0.0, June 2009. 4

[2] A. Agarwala. Efficient gradient-domain compositing using quadtrees. *ACM Transactions on Graphics*, 26(3), August 2007. 1, 2, 3, 6, 7

[3] A. Agarwala, M. Dontcheva, M. Agrawala, S. Drucker, A. Colburn, B. Curless, D. H. Salesin, and M. F. Cohen. Interactive digital photomontage. *ACM Transactions on Graphics (Proc. SIGGRAPH 2004)*, 23(3):292–300, August 2004. 1, 2

[4] P. Bhat, B. Curless, M. Cohen, and C. L. Zitnick. Fourier analysis of the 2D screened Poisson equation for gradient domain problems. In *Tenth European Conference on Computer Vision (ECCV 2008)*, pages 114–128. Springer-Verlag, October 2008. 1

[5] P. Bhat, C. L. Zitnick, M. F. Cohen, and B. Curless. Gradientshop: A gradient-domain optimization framework for image and video filtering. *ACM Transactions on Graphics*, 29(2), March 2010. 2

[6] J. Bolz, I. Farmer, E. Grinspun, and P. Schröder. Sparse matrix solvers on the GPU: Conjugate gradients and multigrid.

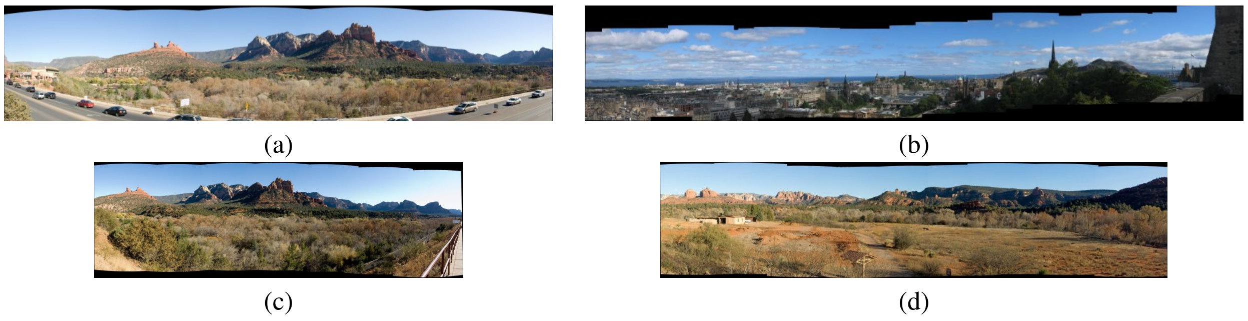

Figure 6. Thumbnails of the four large panorama test images: (a) Sedona, (b) Edinburgh, (c) Crag, (d) RedRock.

ACM Transactions on Graphics (Proc. SIGGRAPH 2003), 22(3):917–924, July 2003. 1

[7] M. Brown and D. Lowe. Automatic panoramic image stitching using invariant features. *International Journal of Computer Vision*, 74(1):59–73, August 2007. 5

[8] L. Buatois, G. Caumon, and B. Lévy. Concurrent number cruncher: An efficient sparse linear solver on the gpu. In *High Performance Computing Conference (HPCC 2007)*, pages 358–371. Springer-Verlag, 2007. 1

[9] P. J. Burt and E. H. Adelson. A multiresolution spline with applications to image mosaics. *ACM Transactions on Graphics*, 2(4):217–236, October 1983. 1, 5

[10] T. A. Davis. *Direct Methods for Sparse Linear Systems*. SIAM, 2006. 6

[11] A. Eden, M. Uyttendaele, and R. Szeliski. Seamless image stitching of scenes with large motions and exposure differences. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2006)*, volume 3, pages 2498–2505, June 2006. 1, 4

[12] Z. Farbman, G. Hoffer, Y. Lipman, D. Cohen-Or, and D. Lischinski. Coordinates for instant image cloning. *ACM Transactions on Graphics*, 28(3), August 2009. 1, 3

[13] G. E. Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*. Academic Press, Boston, Massachusetts, 4th edition, 1996. 3

[14] R. Fattal, D. Lischinski, and M. Werman. Gradient domain high dynamic range compression. *ACM Transactions on Graphics (Proc. SIGGRAPH 2002)*, 21(3):249–256, 2002. 2

[15] D. B. Goldman and J.-H. Chen. Vignette and exposure calibration and compensation. In *Tenth International Conference on Computer Vision (ICCV 2005)*, volume 1, pages 899–906, October 2005. 1, 4

[16] J. Hays and A. A. Efros. Scene completion using millions of photographs. *ACM Transactions on Graphics*, 26(3), August 2007. 1

[17] J. Jia, J. Sun, C.-K. Tang, and H.-Y. Shum. Drag-and-drop pasting. *ACM Transactions on Graphics*, 25(3):631–636, August 2006. 1

[18] M. Kazhdan and H. Hoppe. Streaming multigrid for gradient-domain operations on large images. *ACM Transactions on Graphics*, 27(3), August 2008. 1, 4

[19] J. Kopf, M. F. Cohen, D. Lischinski, and M. Uyttendaele. Joint bilateral upsampling. *ACM Transactions on Graphics*, 26(3), August 2007. 1, 4

[20] J. Kopf, M. Uyttendaele, O. Deussen, and M. F. Cohen. Capturing and viewing gigapixel images. *ACM Transactions on Graphics*, 26(3), August 2007. 5, 6

[21] A. Levin, D. Lischinski, and Y. Weiss. Colorization using optimization. *ACM Transactions on Graphics*, 23(3):689–694, August 2004. 1

[22] A. Levin, A. Zomet, S. Peleg, and Y. Weiss. Seamless image stitching in the gradient domain. In *Eighth European Conference on Computer Vision (ECCV 2004)*, volume IV, pages 377–389. Springer-Verlag, May 2004. 1

[23] J. McCann and N. S. Pollard. Real-time gradient-domain painting. *ACM Transactions on Graphics*, 27(3), August 2008. 1

[24] P. Pérez, M. Gangnet, and A. Blake. Poisson image editing. *ACM Transactions on Graphics (Proc. SIGGRAPH 2003)*, 22(3):313–318, July 2003. 1, 2, 3

[25] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, second edition, 2003. 6

[26] H.-Y. Shum and R. Szeliski. Construction of panoramic mosaics with global and local alignment. *International Journal of Computer Vision*, 36(2):101–130, February 2000. Erratum published July 2002, 48(2):151–152. 7

[27] R. Szeliski. Locally adapted hierarchical basis preconditioning. *ACM Transactions on Graphics (Proc. SIGGRAPH 2006)*, 25(3):1135–1143, August 2006. 1, 2, 4, 7

[28] R. Szeliski and J. Coughlan. Spline-based image registration. *International Journal of Computer Vision*, 22(3):199–218, March/April 1997. 4

[29] R. Szeliski, M. Uyttendaele, and D. Steedly. Fast Poisson blending using multi-splines. Technical Report MSR-TR-2008-58, Microsoft Research, April 2008. 2, 5, 6

[30] D. Terzopoulos. Multilevel computational processes for visual surface reconstruction. *Computer Vision, Graphics, and Image Processing*, 24:52–96, 1983. 4

[31] K. Van Leemput, F. Maes, D. Vandermeulen, and P. Suetens. Automated model-based bias field correction of MR images of the brain. *IEEE Transactions on Medical Imaging*, 18(10):885–896, 1999. 4