Fast, Robust Image Registration for Compositing High Dynamic Range Photographs from Handheld Exposures

Greg Ward
Exponent – Failure Analysis Assoc.
Menlo Park, CA

Abstract

In this paper, we present a fast, robust, and completely automatic method for translational alignment of hand-held photographs. The technique employs percentile threshold bitmaps to accelerate image operations and avoid problems with the varying exposure levels used in high dynamic range (HDR) photography. An image pyramid is constructed from grayscale versions of each exposure, and these are converted to bitmaps which are then aligned horizontally and vertically using inexpensive shift and difference operations over each image. The cost of the algorithm is linear with respect to the number of pixels and effectively independent of the maximum translation. A three million pixel exposure can be aligned in a fraction of a second on a contemporary microprocessor using this technique.

1. Introduction

Although there are a few cameras entering the commercial market that are capable of direct capture of high dynamic-range (HDR) photographs, most researchers still employ a standard digital or analog camera and composite HDR images from multiple exposures using the technique pioneered by Debevec and Malik [Debevec97]. One of the chief limitations of this technique is its requirement that the camera be absolutely still between exposures. Even tripod mounting will sometimes allow slight shifts in the camera's position that yield blurry or double images. If there were a fast and reliable method to align image exposures, one would be able to take even handheld photos and reconstruct an HDR image. Since many of the mid-priced digital cameras on the consumer market have a bracketed exposure mode, it is quite easy to take three to five exposures with different speed settings, suitable for HDR reconstruction. After taking 50 or so handheld exposure sequences ourselves (each having five exposures), we made a number of unsuccessful attempts to reconstruct HDR images using available tools and methods, and decided to develop a new approach.

After experimenting with a few different algorithms, some borrowed from others and some developed ourselves, we found the bitmap method described here, which is very efficient at determining the optimal pixel-resolution alignment between exposures. The basic idea behind our approach is simple, and we explain this along with the more subtle points not to be neglected in Section 2. Some of the results and ideas for future work are presented in Section 3, followed by a brief conclusion.

2. Method

Input to our alignment algorithm is a series of N 8-bit grayscale images, which may be approximated using only the green channel, or better approximated from 24-bit sRGB using the formula below.[1]

$$\text{grey} = (54*\text{red} + 183*\text{green} + 19*\text{blue}) / 256$$

One of the N images is arbitrarily selected as the reference image, and the output of the algorithm is a series of N-1 (x,y) integer offsets for each of the remaining images relative to this reference. These exposures may then be recombined efficiently into an HDR image using the camera response function, which may be computed using either Debevec and Malik's original SVD technique [Debevec97], or as we have done in this paper, using the polynomial method of Mitsunaga and Nayar [Mitsunaga99].

We focused our computation on integer pixel offsets, because they can be used to quickly recombine the exposures without resampling. Also, we found that 90% of the handheld sequences we took did not require rotational alignment. Even in sequences where there was some discernable rotation, the effect of a good translational alignment was to push the blurred pixels out to the edges, where they are less distracting to the viewer. Rotation and subpixel alignment are interesting problems in their own right, and we discuss them briefly at the end of the paper.

We found out the hard way that conventional approaches to image alignment usually fail when applied to images with large exposure variations. In general, edge-detection filters are dependent on image exposure, as shown in the left side of Figure 1, where edges appear and disappear at different exposure levels. Edge-matching algorithms are therefore ill-suited to the exposure alignment problem. We tried the Skeleton Subspace Deformation (SSD) code of Hector Yee [Yee01], which took a considerable length of time to find and align edges, and failed on the majority of our high-resolution examples. Similarly, we were frustrated in our efforts to find a point-feature matching algorithm that could reliably identify exposure-invariant features in our bracketed sequences [Cheng96].

[1] We have not found the red and blue channels to make any difference to the alignment results, so we just use the green channel.

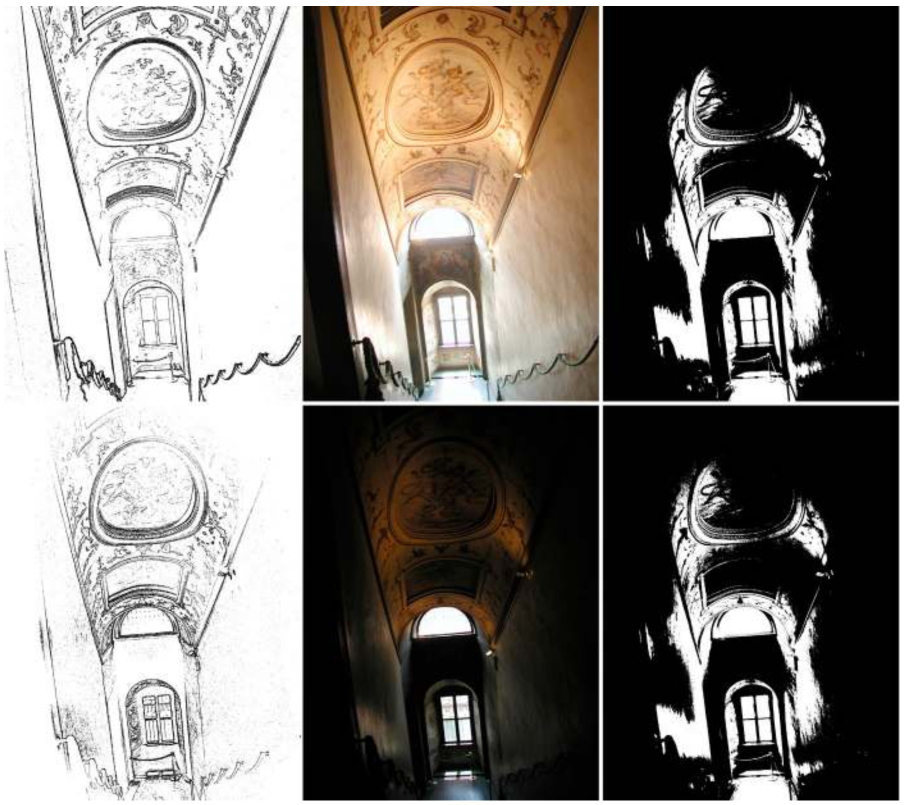

Figure 1. Two unaligned exposures (middle) and their corresponding edge bitmaps (left) and median threshold bitmaps (right). The edge bitmaps are not used in our algorithm, precisely because of their tendency to shift dramatically from one exposure level to another. In contrast, the MTB is stable with respect to exposure.

The alternate approach we describe in this paper has the following desirable features:

- Alignment is done on bi-level images using fast, bit-manipulation routines
- Alignment is insensitive to image exposure
- Alignment includes noise filtering for robustness

The results of a typical alignment are shown in Figure 6 in the results section.

If we are to rely on operations such as moving, multiplying, and subtracting pixels over an entire high-resolution image, our algorithm is bound to be computationally expensive, unless our operations are very fast. Bitmap images allow us to operate on 32 or 64 pixels at a time using bitwise integer operations, which are very fast compared to byte-wise arithmetic. We experimented with a few different binary image representations until we found one that facilitated image alignment independent of exposure level: the *median threshold bitmap*.

A median threshold bitmap (MTB) is defined as follows:

1. Determine the median 8-bit value from a low-resolution histogram over the grayscale image pixels.
2. Create a bitmap image with 0's where the input pixels are less than or equal to the median value and 1's where the pixels are greater.

Figure 1 shows two exposures of an Italian stairwell in the middle and their corresponding edge maps on the left and MTBs on the right. In contrast to the edge maps, our MTBs are nearly identical for the two exposures. Taking the difference of these two bitmaps with an exclusive-or (XOR) operator shows where the two images are misaligned, and small adjustments in the x and y offsets yield predictable changes in this difference due to object coherence. However, this is not the case for the edge maps, which are noticeably different for the two exposures, even though we attempted to compensate for the camera response with an approximate response curve. Taking the difference of the two edge bitmaps would not give a good indication of where the edges are misaligned, and small changes in the x and y offsets yield unpredictable results, making gradient search problematic. More sophisticated methods to determine edge correspondence are necessary to use this information, and we can avoid these and their associated computational costs with our MTB-based technique.

The constancy of an MTB with respect to exposure is a very desirable property for determining image alignment. For most HDR reconstruction algorithms, the alignment step must be completed before the camera response can be determined, since the response function is derived from corresponding pixels in the different exposures. An HDR alignment algorithm that depends on the camera response function would thus create a "chicken and egg" problem. By its nature, an MTB is the same for any exposure within the usable range of the camera, regardless of the response curve. So long as the camera's response function is monotonic with respect to world radiance, the same scene will theoretically produce the same MTB at any exposure level. This is because the MTB partitions the pixels into two equal populations, one brighter and one darker than the scene's median value. Since the median value does not change in a static scene, our derived bitmaps likewise do not change with exposure level.[2]

There may be certain exposure pairs that are either too light or too dark to use the median value as a threshold without suffering from noise, and for these we choose either the 17th or 83rd percentile as the threshold, respectively. Although our offset results are all relative to a designated reference exposure, we actually compute offsets between adjacent exposures, so the same threshold may be applied to both images. Choosing percentiles other than the 50th (median) results in fewer pixels to compare, and this makes the solution less stable, so we may choose to limit the maximum offset in certain cases. The behavior of percentile threshold bitmaps is otherwise the same as the MTB, including stability over different exposures. In the remainder of the paper, when we refer to the properties and operations of MTBs, you can assume that the same applies for other percentile threshold bitmaps as well.

Once we have our threshold bitmaps corresponding to the two exposures we wish to align, there are a number of ways to go about aligning them. One brute force approach is to test every offset within the allowed range, computing the XOR difference at each offset and taking the coordinate pair corresponding to the minimum difference. A more efficient approach might follow a gradient descent to a local minimum, computing only

[2] Technically, the median value could change with changing boundaries as the camera moves, but such small changes in the median are usually swamped by noise, which is removed by our algorithm as we will explain.

local bitmaps differences between the starting offset (0,0) and the nearest minimum. We choose a third method based on an image pyramid that is as fast as gradient descent in most cases, but is more likely to find the global minimum within the allowed offset range.

Multi-scale techniques are well-known in the computer vision and image-processing communities, and image pyramids are frequently used for registration and alignment. (See for example [Thevenaz98].) In our technique, we start by computing an image pyramid for each grayscale image exposure, with $\log_2$(max_offset) levels past the base resolution. The resulting MTBs are shown for our two example exposures in Figure 2. For each smaller level in the pyramid, we take the previous grayscale image and filter it down by a factor of two in each dimension, computing the MTB from the grayscale result.[3]

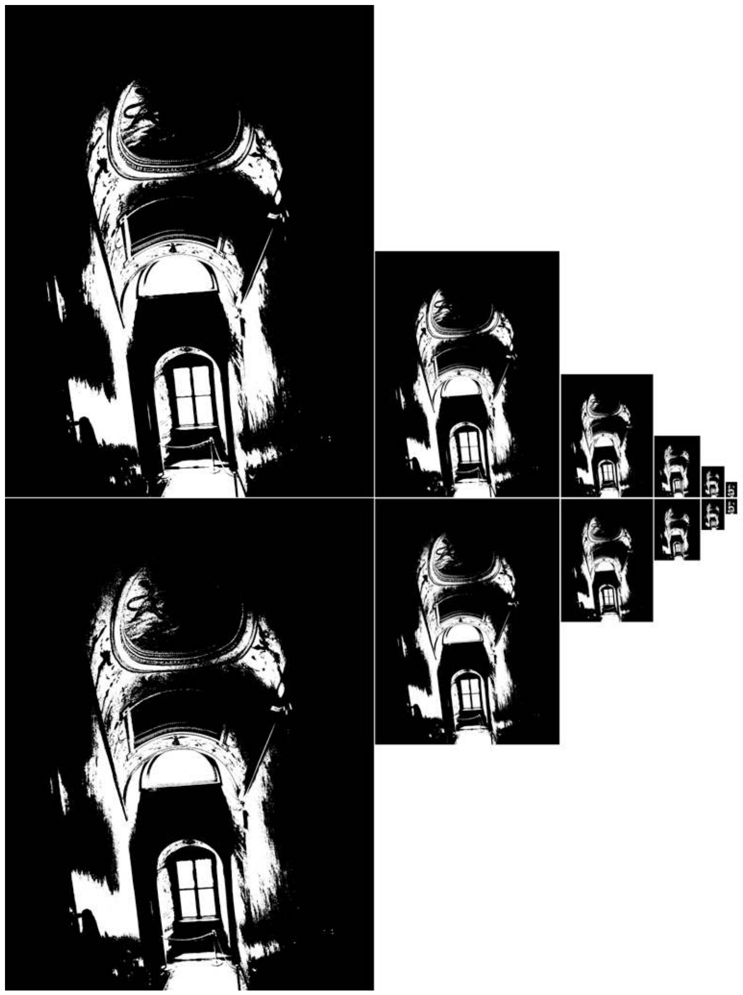

Figure 2. A pyramid of MTBs are used to align adjacent exposures one bit at a time. The smallest (rightmost) image pair corresponds to the most significant bit in the final offset.

[3] Be careful *not* to subsample the bitmaps themselves, as the result will be subtley different and could potentially cause the algorithm to fail.

To compute the overall offset for alignment, we start with the lowest resolution MTB pair and compute the minimum difference offset between them within a range of ±1 pixel in each dimension. At the next resolution level, we multiply this offset by 2 (corresponding to the change in resolution) and compute the minimum difference offset within a ±1 pixel range of this previous offset. This continues to the highest (original) resolution MTB, where we get our final offset result. Thus, each level in the pyramid corresponds to a binary bit in the computed offset value.

At each level, we need to compare exactly 9 candidate MTB offsets, and the cost of this comparison is proportional to the size of the bitmaps. The total time required for alignment is thus linear with respect to the original image resolution and independent of the maximum offset, since our registration step is linear in the number of pixels, and the additional pixels in an image pyramid are determined by the size of the source image and the (fixed) height of the pyramid.

Threshold Noise

The algorithm just described works well in images that have a fairly bimodal brightness distribution, but can run into trouble for exposures that have a large number of pixels near the median value. In such cases, the noise in near-median pixels shows up as noise in the MTB, which destabilizes our difference computations.

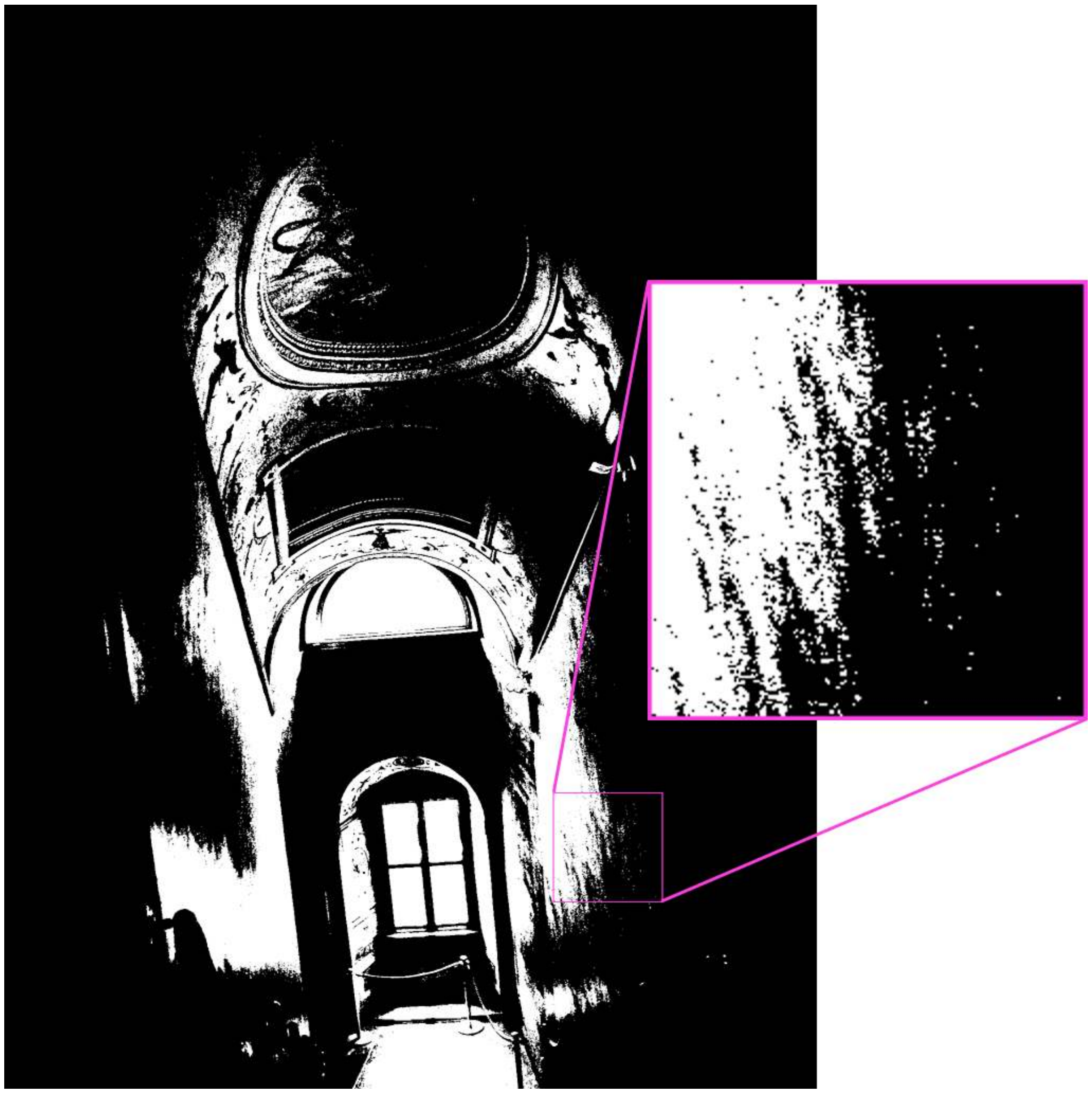

Figure 3. Close-up detail of noisy area of MTB in dark stairwell exposure (full resolution).

The inset in Figure 3 shows a close-up of the pixels in our dark stairwell exposure MTB, and the kind of noise we see in some images. Computing the XOR difference between exposures with large areas like these yields noisy results that are unstable with respect to translation, because the pixels themselves tend to move around in different exposures. Fortunately, there is a straightforward solution to this problem.

Since our problem involves pixels whose values are close to our threshold, we can exclude these pixels from our difference calculation with an *exclusion bitmap*. Our exclusion bitmap consists of 0's wherever the grayscale value is within some specified distance of the threshold, and 1's elsewhere. The exclusion bitmap for the exposure in Figure 3 is shown in Figure 4, where we have zeroed all bits where pixels are within ±4 of the median value.

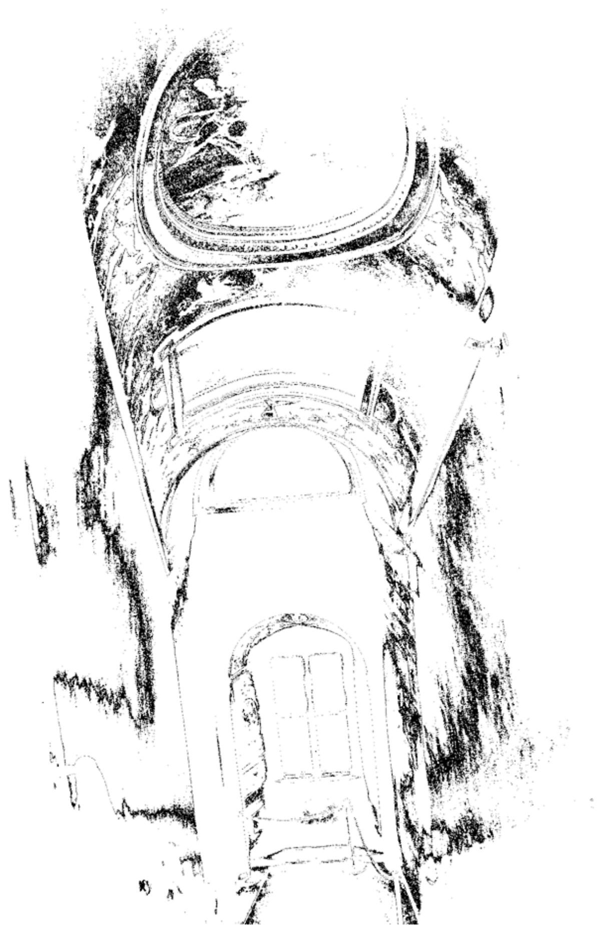

Figure 4. An exclusion bitmap, with zeroes (black) wherever pixels in our original image are within the noise tolerance of the median value.

We compute an exclusion bitmap for each exposure at each resolution level in our pyramid, then take the XOR difference result for our candidate offset, AND'ing it with both offset exclusion bitmaps to compute our final difference.[4] The effect is to disregard differences that are less than the noise tolerance in our images. This is illustrated in Figure 5, where we see the XOR difference of the unaligned exposures before and after applying the exclusion bitmaps. By removing those pixels that are close to the median, we clear the least reliable bit positions in the smooth gradients, but preserve the high confidence pixels near strong boundaries, such as the edges of the window and doorway. Empirically, we have found this optimization to be very effective in eliminating false minima in our offset search algorithm.

[4] If we were to AND the exclusion bitmaps with the original MTBs before the XOR operation, we would inadvertantly count disagreements about what was noise and what was not as actual pixel differences.

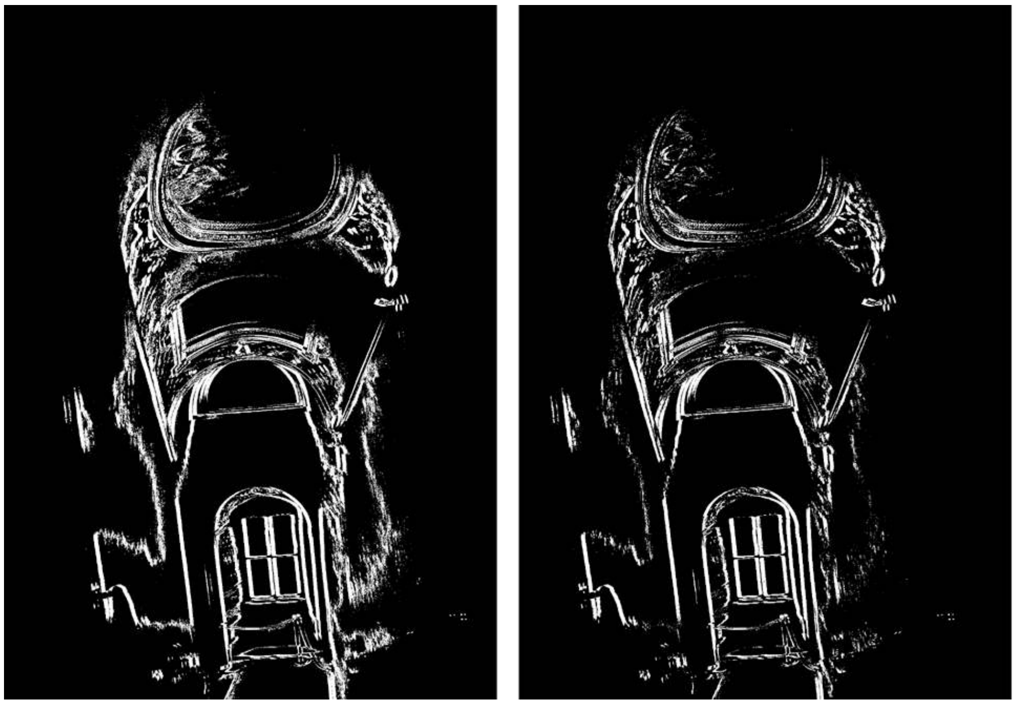

Figure 5. The original XOR difference of our unaligned exposures (left), and with the two exclusion bitmaps AND'ed into the result to reduce noise in the comparison (right).

Overall Algorithm

The overall algorithm with the exclusion operator is given in the following recursive C function, `GetExpShift`. This function takes two exposure images, and determines how much to move the second exposure (img2) in x and y to align it with the first exposure (img1). The maximum number of bits in the final offsets is determined by the shift_bits parameter. We list the more important functions called by `GetExpShift` in Table 1, and leave their implementation as an exercise for the reader.

`ImageShrink2(const Image *img, Image *img_ret)`	Subsample the image img by a factor of two in each dimension and put the result into a newly allocated image img_ret.
`ComputeBitmaps(const Image *img, Bitmap *tb, Bitmap *eb)`	Allocate and compute the threshold bitmap tb and the exclusion bitmap eb for the image img. (The threshold and tolerance to use are included in the Image struct.)
`BitmapShift(const Bitmap *bm, int xo, int yo, Bitmap *bm_ret)`	Shift a bitmap by (xo,yo) and put the result into the preallocated bitmap bm_ret, clearing exposed border areas to zero.
`BitmapXOR(const Bitmap *bm1, const Bitmap *bm2, Bitmap *bm_ret)`	Compute the "exclusive-or" of bm1 and bm2 and put the result into bm_ret.
`BitmapTotal(const Bitmap *bm)`	Compute the sum of all 1 bits in the bitmap.

Table 1. Functions called by `GetExpShift`.

```
GetExpShift(const Image *img1, const Image *img2, int shift_bits, int
shift_ret[2])
{
      int         min_err;
      int         cur_shift[2];
      Bitmap      tb1, tb2;
      Bitmap      eb1, eb2;
      int   i, j;
      if (shift_bits > 0) {
            Image       sml_img1, sml_img2;
            ImageShrink2(img1, &sml_img1);
            ImageShrink2(img2, &sml_img2);
            GetExpShift(&sml_img1, &sml_img2, shift_bits-1, cur_shift);
            ImageFree(&sml_img1);
            ImageFree(&sml_img2);
            cur_shift[0] *= 2;
            cur_shift[1] *= 2;
      } else
            cur_shift[0] = cur_shift[1] = 0;
      ComputeBitmaps(img1, &tb1, &eb1);
      ComputeBitmaps(img2, &tb2, &eb2);
      min_err = img1->xres * img1->yres;
      for (i = -1; i < = 1; i++)
            for (j = -1; j <= 1; j++) {
                  int         xs = cur_shift[0] + i;
                  int         ys = cur_shift[1] + j;
                  Bitmap      shifted_tb2;
                  Bitmap      shifted_eb2;
                  Bitmap      diff_b;
                  int         err;
                  BitmapNew(img1->xres, img1->yres, &shifted_tb2);
                  BitmapNew(img1->xres, img1->yres, &shifted_eb2);
                  BitmapNew(img1->xres, img1->yres, &diff_b);
                  BitmapShift(&tb2, xs, ys, &shifted_tb2);
                  BitmapShift(&eb2, xs, ys, &shifted_eb2);
                  BitmapXOR(&tb1, &shifted_tb2, &diff_b);
                  BitmapAND(&diff_b, &eb1, &diff_b);
                  BitmapAND(&diff_b, &shifted_eb2, &diff_b);
                  err = BitmapTotal(&diff_b);
                  if (err < min_err) {
                        shift_ret[0] = xs;
                        shift_ret[1] = ys;
                        min_err = err;
                  }
                  BitmapFree(&shifted_tb2);
                  BitmapFree(&shifted_eb2);
            }
      BitmapFree(&tb1); BitmapFree(&eb1);
      BitmapFree(&tb2); BitmapFree(&eb2);
}
```

Computing the alignment offset between two adjacent exposures is simply a matter of calling the `GetExpShift` routine with the two image struct's (img1 and img2), which contain their respective threshold and tolerance values. (The threshold values must correspond to the same population percentiles in the two exposures.) We also specify the maximum number of bits allowed in our returned offset, shift_bits. The shift results computed and returned in shift_ret will thus be restricted to a range of $\pm 2^{\text{shift\_bits}}$.

There is only one subtle point in the above algorithm, which is what happens at the image boundaries. If we aren't careful, we might inadvertently shift non-zero bits into our candidate image, and count these as differences in the two exposures, which would be a mistake. That is why we need the `BitmapShift` function to shift 0's into the new image areas, so that applying the shifted exclusion bitmap to our XOR difference will clear these exposed edge pixels as well. This also explains why we need to limit the maximum shift offset, because if we allow this to be unbounded, then our lowest difference solution will also have the least pixels in common between the two exposures – one exposure will end up shifted completely off the other. In practice, we have found a shift_bits limit of 6 (±64 pixels) to work fairly well most of the time.

Efficiency Considerations

Clearly, the efficiency of our overall algorithm is going to depend on the efficiency of our bitmap operations, as we perform 9 shift tests with 6 whole-image bitmap operations apiece. The `BitmapXOR` and `BitmapAND` operations are easy enough to implement, as we simply apply bitwise operations on 32-bit or 64-bit words, but the `BitmapShift` and `BitmapTotal` operators may not be so obvious.

For the `BitmapShift` operator, we start with the observation that any two-dimensional shift in a bitmap image can be reduced to a one-dimensional shift in the underlying bits, accompanied by a clear operation on one or two edges for the exposed borders. Implementing a one-dimensional shift of a bit array requires at most a left or right shift of B bits per word with a reassignment of the underlying word positions. Clearing the borders then requires clearing words where sequences of 32 or 64 bits are contiguous, and partial clears of the remainder words. The overall cost of this operator, although greater than the XOR or AND operators, is still modest. Our own `BitmapShift` implementation includes an additional Boolean parameter that turns off border clearing. This allows us to optimize the shifting of the threshold bitmaps, which have their borders cleared later by the exclusion bitmap, and don't need the `BitmapShift` operator to clear them.

For the `BitmapTotal` operator, we precompute a table of 256 integers corresponding to the number of 1 bits in the binary values from 0 to 255 (i.e., {0, 1, 1, 2, 1, 2, 2, 3, 1, …, 8}). We then break each word of our bitmap into byte-sized chunks, and add together the corresponding bit counts from the precomputed table. This results in a speed-up of at least 8 times over counting individual bits, and may be further accelerated by special-case checking for zero words, which occur frequently in our application.

The author has implemented the above operations in a bitmap class using C++, which we are happy to make available to anyone who sends us a request for the source code.

3. Results and Discussion

Figure 6 shows the results of applying our image alignment algorithm to all five exposures of the Italian stairwell, with detail close-ups showing before and after alignment. The misalignment shown is typical of a handheld exposure sequence, requiring translation of several pixels on average to bring the exposures back atop each other. We have found that even tripod exposures sometimes need minor adjustments of a few pixels for optimal results.

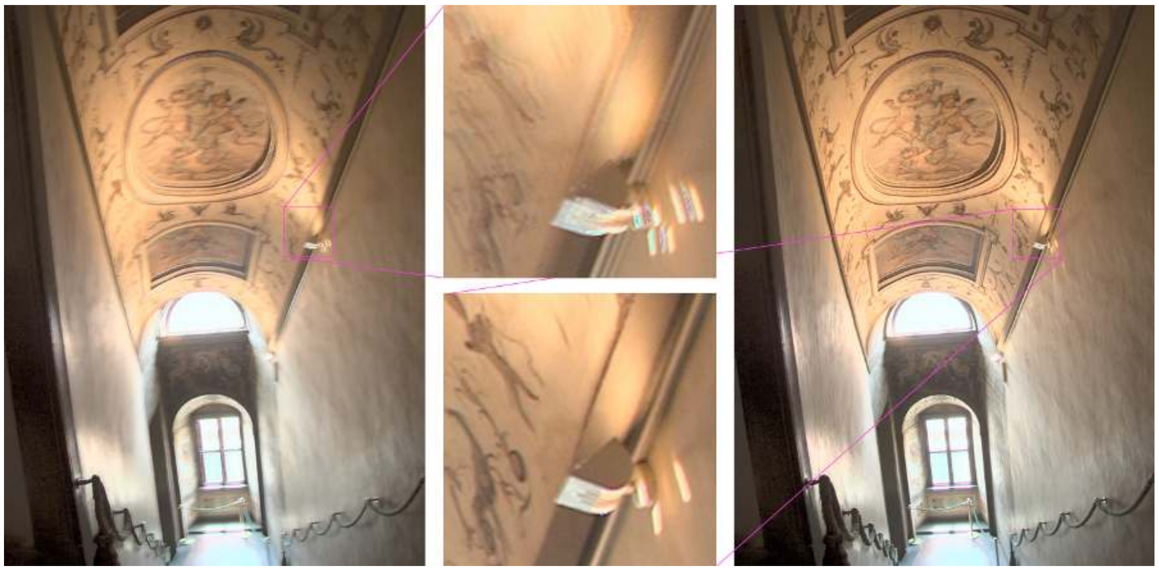

Figure 6. An HDR image composited from unaligned exposures (left) and detail (top center). Exposures aligned with our algorithm yield a superior composite (right) with clear details (bottom center).

We have applied our simple translational alignment algorithm to over 100 hand-held exposure sequences. Overall, our success rate has been about 84%, with 10% giving unsatisfactory results due to image rotation. About 3% of our sequences failed due to excessive scene motion – usually waves or ripples on water that happened to be near the threshold value and moved between frames, and another 3% had too much high frequency content, which made the MTB correspondences unstable.[5] Most of the rotation failures were mild, leaving at least a portion of the HDR image well-aligned. Other failures were more dramatic, throwing alignment off to the point where it was better not to apply any translation at all. It may be possible to detect such failure cases by looking at the statistics of MTB differences in successful and unsuccessful alignments, but we have not tested this idea as yet.

Although exposure rotation is less frequent and less objectionable than translation, it happens often enough that automatic correction is desirable. One possible approach is to apply our alignment algorithm separately to each quadrant of the image. If the computed translations are the same or nearly so, then no rotation is applied. If there are significant differences attributable to rotation, we could apply a rotation-filtering algorithm to arrive at an aligned result. This would be more expensive than straight image translation, but since it happens in relatively few sequences, the average cost might be acceptable. The divide-and-conquer approach might also enable us to detect inconsistent offset results, which would be a sign that our algorithm was not performing correctly.

[5] We thought we might correct some of these alignments if we allowed a ±2 pixel shift at each level of the pyramid rather than ±1, but our tests showed no improvement.

If we are willing to accept the filtering expense associated with rotational alignment, we may as well align images at a subpixel level. To accomplish this, the MTB algorithm could be extended so that one or two levels above the highest resolution bitmap are generated via bilinear or bicubic interpolation on the original image. Such a representation would be needed for rotation anyway, so the additional expense of this approach would be nominal.

Ultimately, we would like digital camera manufacturers to incorporate HDR capture into their products. Although the algorithm we describe has potential for onboard image alignment due to its speed and small memory footprint, it would be better if the exposures were taken close enough together that camera motion was negligible. The real problem is in establishing a compressed, high dynamic range image format that the camera manufacturers and imaging software companies can support. We have proposed two standards for encoding high dynamic range colors [Ward91] [Larson98], but work still needs to be done to incorporate these color spaces into an image compression standard, such as JPEG 2000 [Christopoulos2000].

Conclusion

We have presented a fast, robust method for aligning hand-held photographs taken at different exposure levels for the purpose of high dynamic range image composition. The method is based on a pyramid of median threshold bitmaps, which are aligned using bitwise shift and differencing operations. The technique requires less than a second on a contemporary microprocessor to align a three million pixel exposure, and is linear in time and storage with the number of pixels. This is comparable to the time required to composite the exposure into an HDR result once its alignment is known.

Future work may include methods for detecting when our alignment algorithm has failed, and techniques for determining and applying rotation and subpixel alignment.

References

[Cheng96] Yong-Qing Cheng, Victor Wu, Robert T. Collins, Allen R. Hanson, and Edward M. Riseman, "Maximum-Weight Bipartite Matching Technique and Its Application in Image Feature Matching," SPIE Conference on Visual Communication and Image Processing, Orlando, FL, 1996.

[Christopoulos2000] C. Christopoulos, A. Skodras, and T. Ebrahimi. "The JPEG2000 Still Image Coding: An Overview," *IEEE Transactions on Consumer Electronics*, Vol. 46, No. 4, pp. 1103-1127, November 2000.

[Debevec97] Paul Debevec, Jitendra Malik, Recovering High Dynamic Range Radiance Maps from Photographs, *Computer Graphics (Proceedings of SIGGRAPH 97)*, ACM, 1997.

[Larson98] Greg Ward Larson, "LogLuv encoding for full-gamut, high-dynamic range images," *Journal of Graphics Tools*, 3(1):15-31, 1998.

[Mitsunaga99]
T. Mitsunaga and S. K. Nayar, “Radiometric Self Calibration,” *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, Fort Collins, June, 1999.

[Thevenaz 98]
P. Thevenaz, U. E. Ruttimann, and M. Unser, “A pyramid approach to subpixel registration based on intensity,” *IEEE Transactions on Image Processing*, vol. 7, No. 1, January, 1998.

[Ward91] Greg Ward, “Real Pixels,” *Graphics Gems II*, edited by James Arvo, Academic Press, 1991.

[Yee01] Personal communication <yeehector@hotmail.com>, Sept. 2001.