Display-aware Image Editing

Anonymous ICCP submission

Paper ID ****

Abstract

We describe a set of image editing and viewing tools that explicitly take into account the resolution of the display on which the image is viewed. Our approach is twofolds. First, we design editing tools that process only the visible data, which is useful for images larger than the display. This encompasses cases such as multi-image panoramas and high-resolution medical data. Second, we propose an adaptive way to set viewing parameters such brightness and contrast. Because we deal with very large images, different locations and scales often require different viewing parameters. We let users set these parameters at a few places and interpolate satisfying values everywhere else. We demonstrate the efficiency of our approach on different display and image sizes. Since the computational complexity to render a view depends on the display resolution and not the actual input image resolution, we achieve interactive image editing even on a 16 gigapixel image.

1. Introduction

Gigapixel images are now commonplace with dedicated devices to automate the image capture [2, 1, 37, 24] and image stitching software [5, 12]. These large pictures have a unique appeal compared to normal-sized images. Fully zoomed out, they convey a global sense of the scene, while zooming in lets one dive in, revealing the smallest details, as if one were there. In addition, modern scientific instruments such as electron microscopes or sky-surveying telescopes are able to generate very high-resolution images for scientific discovery at the nano- as well as at the cosmological scale. We are interested in two problems related to these large images: editing them and viewing them. Editing such large pictures remains a painstaking task. Although after-exposure retouching plays a major role in the rendition of a photo [3], and enhancing scientific images is critical to their interpretation [9], these operations are still mostly out of reach for images above 100 megapixels. Standard editing techniques are designed to process images that have at most a few megapixels. While significant speed ups have been obtained at these intermediate resolutions, e.g. [15, 17, 18, 28], major hurdles remain to interactively edit larger images. For instance, optimization tools such as least-squares systems and graph cuts become unpractical when the number of unknowns approaches or exceeds a billion. Furthermore, even simple editing operations become costly when repeated for hundreds of millions of pixels. The basic insight of our approach is that the image is viewed on a display with a limited resolution, and only a subset of the image is visible at any given time. We describe a series of image editing operators that produce results only for the visible portion of the image and at the displayed resolution. A simple and efficient multi-resolution data representation (§ 2) allows each image operator to quickly access the currently visible pixels. Because the displayed view is computed on demand, our operators are based on efficient image pyramid manipulations and designed to be highly data parallel (§ 3). When the user changes the view location or resolution we simply recompute the result on the fly.

Further, editing tools do not support the fact that very large images can be seen at multiple scales. For instance, a high-resolution scan as shown in the companion video reveals both the overall structure of the brain as well as the fine entanglement between neurons. In existing software, settings such as brightness and contrast are the same, whether one looks at the whole image or at a small region. In comparison, we let the user specify several viewing settings for different locations and scales. This is useful for emphasizing different structures, e.g. on a brain scan, or expressing different artistic intents, e.g. on a photo. We describe an interpolation scheme motivated by a user study to infer the viewing parameters where the user has not specified any settings (§ 4). This adaptive approach enables the user to obtain a pleasing rendition at all zoom levels and locations while setting viewing parameters only at a few places.

The novel contributions of this work are twofolds. First, we describe editing operators such as image stitching and seamless cloning that are output-sensitive, i.e., the associated computational effort depends only on the display resolution. Our algorithms are based on Laplacian pyramids, which we motivate by a theoretical study of the constraints required to be display-aware. Second, we propose an interpolation scheme motivated by a user study to infer viewing parameters where the user has not specified any settings. We illustrate our approach with a prototype implemented on the GPU and show that we can interactively edit very large images as large as 16 gigapixels.

1.1. Related Work

The traditional strategy with large images is to process them at full resolution and then rescale and crop according to the current display. As far as we know, this is commonly used in commercial software. However, this simple approach becomes quickly unpractical with large images, especially with optimization such as graph cuts and Poisson solvers.

Fast image filters have been proposed to speed up operations such as edge-aware smoothing [32, 15, 4, 16, 18], seamless compositing [17, 5, 22], inpainting [10], and selection [28]. Although these algorithms reduce the computation times, they have been designed for standard-size images and the entire picture at full resolution is eventually processed. In comparison, we propose display-aware algorithms that work locally in space and scale such that only the visible portion of the data is processed.

Berman et al. [11] and Velho and Perlin [36] describe multi-scale painting systems for large images based on wavelets. From an application perspective, our work is complementary as we do not investigate methods for painting but rather for adaptive viewing and more advanced editing such as seamless cloning. Technically speaking, our methods operate in a display-aware fashion, and not in a multi-scale fashion. That is, we apply our edits on-the-fly to the current view and never actually propagate the results to all scales. Further, it is unclear how to extend the proposed approach from painting to algorithms such as seamless cloning. Pinheiro and Velho [34] and Kopf et al. [24] propose a multi-resolution tiled memory management system for viewing large data. Our data management follows similar design principles, but supports multiple input images that can be aligned to form a local image pyramid on-the-fly without managing a pre-built global multiresolution image pyramid. It also naturally supports out-of-core computations on graphics hardware with limited memory.

Kopf et al. [24] applies a histogram-based tone-mapper to automatically adjust the current view of large HDR images. Our work can also be automatic but also let the user to override the default settings as many times as desired. This allows users to make adjustments that adapt to the current view and may reflect subjective intents. Furthermore, we propose more complex output-sensitive algorithms for tasks such as seamless cloning. Efforts have also been made to develop viewers suitable for multi-layer gigapixel medical data interactively [7]. In comparison, we focus single-layer images and also tackle editing issues.

Shantzis [35] describes a method to limit the amount of computation by only processing the data within the bounding box of each operator. We extend this approach is several ways. Unlike Shantzis, we deal with changes of zoom level and ignore the high-frequency data when they are not visible. This property is nontrivial as we shall see (§ 3.1). We also design new algorithms that enable display-aware processing such as our stitching method based on local computation only. In comparison, the standard method based on graph cut is global, i.e. the bounding box would cover the entire image. Further, we also deal with viewing parameters, which is not in the scope of Shantzis' work.

2. Data Representation

A major aspect of our approach is that the view presented to the user is always computed on the fly. From a data structure point of view, this implies that the displayed pixel data have to be readily available and that we can rapidly determine how to process them. To achieve this, we use several mechanisms detailed in the rest of this section.

2.1. Global Space and Image Tiles

Our approach is organized around a coordinate system in which points are located by their (x, y) spatial location and the scale s at which they are observed. A unit scale $s = 1$ corresponds to the full-resolution data, while $s = \frac{1}{n}$ corresponds to the image downsampled by a factor of n. We use this coordinate system to define *global space* in which we locate data with respect to the displayed image that the user observes (Fig. 1). Typically, we have several input images that make up, e.g., a panorama. For each image I_i, we first compute a geometric transformation g_i that aligns it with the others by specifying its location in global space. If we have only one input image, then g is the identity function. The geometric alignment can either be pre-computed before the editing session, or each image can be aligned on the fly when it is displayed. In the former case, we use feature point detection and homography alignment, e.g. [12]. In the latter case, the user interactively aligns the images in an approximate manner. We then automatically register them in a display-aware fashion by maximizing the cross-correlation between visible overlapping areas. This is useful for images that are being produced on-line by automated scientific instruments. We decompose all input images into *tiles*. For each tile, we pre-compute a Gaussian pyramid to enable access to any portion of the input images at arbitrary resolutions. For resolutions that we have not pre-computed we fetch the pyramid level with a resolution just higher than the requested one and downsample it on the fly. The resampling step is essentially free on graphics hardware, and although we load more data than needed, the overhead is small compared to loading the full-resolution tile or the entire input image. We further discuss the computational complexity of this operation in Section 5.

2.2. Operator Representation

We distinguish two types of operators. *Local operators*, such as copy-and-paste or image cloning, affect only a subset of the image. We store their bounding box in global space as well as an index that indicates in which order the user has performed the edits. We did not include scale s in this representation because we could not conceive of any realistic scenarios in which a local operator would apply only at certain scales, but including it would be straightforward if needed. When the user moves the display to a new position, the viewport defines a display rectangle at a given scale in global space. We test each operator and keep only the ones whose bounding box intersect with the viewport. In our current implementation operators are stored in a list and we test them all since bounding box intersections are efficient. Once we have identified the relevant operators, we apply them in order to the visible pixels at the current resolution. The *global operators* brightness, contrast, and saturation, affect all the pixels. We apply these transformations after the local operators and always in the same order: brightness, contrast, saturation. If the user modifies a setting twice, we keep only the last one. We found that it is beneficial to let users specify different values at different positions and scales. In this case, we store one setting at each (x, y, s) location where the user makes an adjustment and interpolate these values to other locations (§ 4).

3. Local Operators

In this section, we describe local editing operators. The algorithms are designed to be display-aware, that is, we process only the visible portion of the image at the current resolution and perform only a fixed amount of computation per pixel. We first study these operators from a theoretical standpoint and then illustrate our strategy on two specific tasks: seamless cloning and panorama stitching.

3.1. Theoretical Study

We study the requirements that an operator f must satisfy to be display-aware. The function f takes an image I as input and creates an image O as output, that is, $O = f(I)$. To be display-aware, f must be able to compute the visible portion of the output using only the corresponding input data. First, we characterize how the visible portion of an image relates to the full-resolution data. We consider an image X. To be displayed, X is resampled at the screen resolution and cropped. We only consider the case where the screen resolution is lower than the image resolution. The opposite case is only about interpolating pixel values and does not need a special treatment. Downsampling the image X is done with a low-pass filter ℓ followed by a comb filter. Assuming a perfect low-pass filter, ℓ is a multiplication by a box filter in the Fourier domain. After this, the comb filter does not remove any information and we can ignore it. The other effect of displaying the image on a screen is that only part of it is visible. This is a cropping operation c that is a multiplication by a box function in the image domain. We define the operator $s(X) = c(\ell(X))$ that displays X on a screen.

To be display-aware, f must satisfy $s(f(I)) = f(s(I))$, that is, we must be able to compute the visible portion of the output $s(f(I))$ using only the visible portion of the input $s(I)$. A sufficient condition is that f commutes with ℓ and c. ℓ can be any arbitrary box centered in the Fourier domain. To commute with it, f must be such that the content of $f(X)$ at a frequency (u_0, v_0) depends only on the content of X at frequencies $|u| \leq |u_0|$ and $|v| \leq |v_0|$. The rationale is that these frequencies are preserved by ℓ even if its cut-off is (u_0, v_0). The crop function c applies an arbitrary

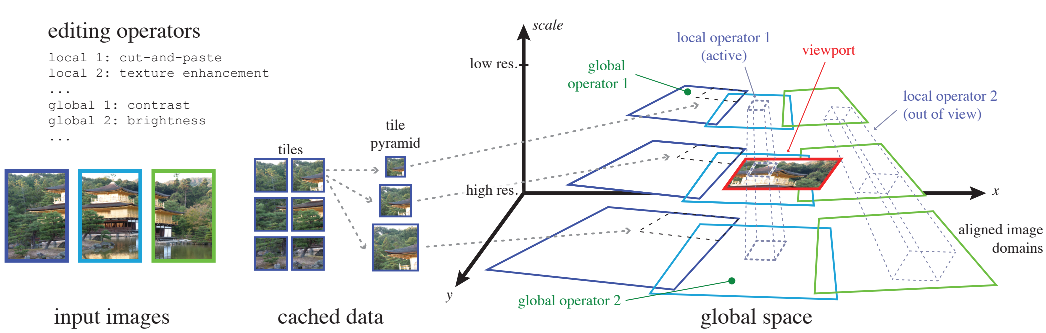


Figure 1. Our approach is based on a caching scheme that ensures that the pixel data are readily available to the editing algorithms. We decompose each input image into tiles and compute a multi-resolution pyramid for each tile. We register the tiles into a common coordinate system, the *global space*. We determine the visible tiles by intersecting them with the viewport rectangle. To the display pixels we either apply *local operators* with a bounding box that intersects the viewport or interpolated *global operators* such as brightness and contrast.

box in image space. For f to commute with it, it must be a point-wise operator since there is no guarantee that adjacent pixels are available. However, these two conditions are too strict to allow for any useful filter. We relax the latter one by considering an "extended screen". For instance, for an operator based on 5×5 windows, add a 2-pixel margin. We apply a similar relaxation in the Fourier domain by adding a "frequency margin", i.e., the input image is resampled at a slightly higher resolution, typically the closest power-of-two resolution. In both cases, the number of processed pixels remain on the same order as the display resolution.

A strategy to satisfy these requirements is to decompose the image I into a Laplacian pyramid and process each level independently and locally. If a process generates out-of-band content, we could post-process the levels to remove this spurious content but we did not find it useful in the examples shown in this paper. This approach yields data-parallel algorithms since constructing a Laplacian pyramid involves purely local operations and so do our display-aware filters.

3.2. On-the-fly Image Alignment and Stitching

Existing large-scale image viewers require a globally aligned and stitched full-resolution panorama to build a multiresolution image pyramid [24, 34]. Poisson compositing is commonly used to stitch multiple images into a panorama [25, 5], but for very large images even optimized methods become costly. Further, recent automated image scanners [21] can produce large images at a speed of up to 11 GB/s. In such a scenario, it is useful to get a quick overview of the entire image with coarse alignment, and to refine the alignment on-the-fly as the user zooms in.

Our on-the-fly alignment assumes that the input images are approximately in the right position in global space. This is the case for automated panorama acquisition systems and scientific instruments. Otherwise, the user can manually align them or run an feature detection algorithm such as [12]. We first adjust the images to have the same exposure and white balance. The affine transformation between images is then automatically refined by maximizing cross-correlation between overlapping regions. We implemented this using gradient descent on the GPU. The alignment is computed for the current zoom level and automatically refined when the user zooms further (see the video, note that in video, refinement is not automatic so that its effect is visible). We stitch the images using the pyramid-based scheme of Burt and Adelson [14]. At each pixel with an overlap, we select the image which border is the farthest, yielding a binary mask for each input I_i. We compute Gaussian pyramids G_i from these masks and Laplacian pyramids L_i from the input images I_i. We linearly blend each level n independently to form a new Laplacian pyramid $\hat{L}^n = \sum_i G_i^n L_i^n / \sum_i G_i^n$. Finally, we collapse the pyramid $\hat{L}$ to obtain the result.

3.3. Push-Pull Image Cloning

Seamless copy-pasting is a standard tool in editing packages [33, 20]. Most implementations rely on solving the Poisson equation and even if optimized algorithms exist [5, 30, 22], this strategy requires to access every pixel at the finest resolution, which does not suit our objectives. Farbman et al. [17] exploit that seamless cloning boils down to smoothly interpolating the color differences at the foreground-background boundary and propose an optimization-free method based on a triangulation of the pasted region. Although it might be possible to adapt Farbman's triangulation to our needs, we propose a pyramid-based method that naturally fits our display-aware context thanks to its multi-scale formulation, and that does not incur the triangulation cost.

We perform a push-pull operation [13] on the color differences at the boundary. We consider a background image B and a foreground image F with a binary mask M. We compute the color offset $O = B - F$ for each pixel on the boundary of M. During the pull phase, we build a Gaussian pyramid from the O values. Since O is only defined at the mask boundary, we ignore all the undefined values during this computation and obtain a sparse pyramid where only some pixels have defined values (and most are empty). Then we collapse the pyramid starting from the coarsest level. In particular, we push pixels with a defined value down to pixels at finer levels that are empty. To avoid blockiness, we employ a bicubic filter during the push phase. This process smoothly fills in the hole [13] and generates an offset map that we add to the foreground pixels before copying them on top of the background.

We apply this process in a display-aware fashion by considering only the visible portion of the boundary. When the user moves the view, appearing and disappearing boundary constraints can induce flickering. Since the offset membrane O is smooth, flickering is only visible near the mask boundary. Thus, we run our process on a slightly extended viewport so that flickering occurs outside the visible region. In practice, we found that extending it by 20 pixels in each direction is enough. Zooming in and out can also cause flickering because the alignment between the boundary and the pyramid pixel grid varies. We address this issue by scaling the data to the next power-of-two, which ensures that the alignment remains consistent.

4. Global Operator Interpolation

For global operators, we have implemented the traditional brightness, contrast, and saturation adjustments. These operators raise specific issues in the context of large images.

(a) Output of Farbman et al. (b) Our result

Figure 2. Although our result and the output of Farbman et al. [17] are not the same, both are satisfying. The input images and Farbman's result come from [17].

Figure 3 shows the difference between an image that is fully zoomed out and fully zoomed in on a shadow region. The same viewing settings cannot be applied to both images. Our solution is adapt the parameters to the location and zoom level. Our approach is inspired by the automatic tone-mapping described by Kopf et al. [24]. Similarly to this technique, our approach can be fully automatic but we also extend it let the user control the settings and offer the possibility to specify different parameters at different locations in the image. We conducted a user study to gain intuition on how to adapt parameters to the current view.

4.1. User Study

We ran a study on Amazon Mechanical Turk where we asked users to adjust the brightness, contrast and saturation of a set of 25 images. The set consisted of 5 crops at various locations and zoom levels from each of 10 different panoramas, for a total of 50 images. We asked the users to adjust the images to obtain a pleasing rendition that was "like a postcard: balanced and vibrant, but not unnatural." Users adjusted brightness, contrast, and saturation, and the initial positions of the sliders were randomized. In total, 27 unique users participated in our study. However, some users made random adjustments to collect the fee. We pruned these results through an independent study, where different users chose between the original and edited images to select which image in the pair was more like a postcard. We kept the results of a given user if his images received at least 65% positive votes. After this, 20 unique users remained.

To analyze a user's edits, we converted the input and output images into the CIE LCH colorspace. As an initial analysis, and inspired by the work on photographic style of Bae et al. [8], we compare the space of lightness histograms before and after editing. We estimate the size of each space by summing the Earth Mover's Distance (EMD) [27] between all pairs of lightness histograms. If the histogram actually characterizes a user's preference, we expect the size of this space to be smaller after the edits. On average, a user's edits reduced the size of the histogram space by 46% compared to the randomized inputs that the user saw, and by 14% compared to the original non-randomized images (not seen by the users), which confirms that the histogram characterizes users' preference. We also analyzed the variance in the distance measurements. We found that all users decreased the variance in histogram distances as compared to the original images. These findings suggest that an interpolation scheme that decreases histogram distances is a good model of user preferences when editing images.

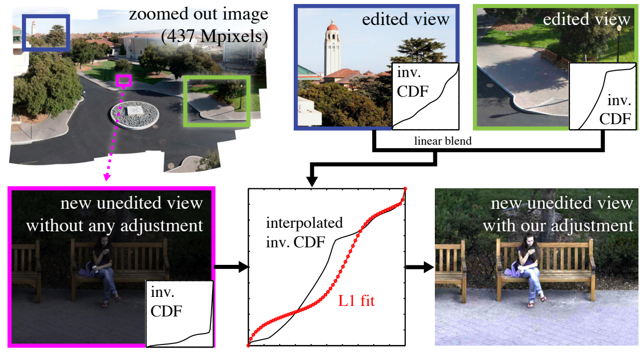


Figure 3. We infer viewing parameters from nearby edits performed by the user. Our scheme linearly interpolates the inverse CDFs of the nearby views and fits brightness and contrast parameters to approximate the interpolated inverse CDF in the L_1 sense.

4.2. Propagation of Edits

The goal of edit propagation is to determine a set of parameters for the current view based on other edits in the image. In the user study, we observed that users tend to make the histograms of images more similar. Accordingly, our approach seeks parameters that make the current histogram close to the histograms of nearby edited regions. The fully zoomed out view always counts as an edited region even if the user keeps the default settings. If the user does not specify any edit, our method is fully automatic akin to the Kopf's viewer [24] and uses the zoomed out view as reference. However, the user can specify edits at any time and out method starts interpolating the user's edits. Let (x_v, y_v, s_v) be the spatial and scale coordinates of the current view. We combine the histograms of the k closest edits into a target histogram. We use the Earth Mover's Distance on the image histograms to find these nearest neighbors. This metric can be interpreted as a simple scene similarity that can be computed efficiently unlike more complex methods [31]. Drawing from work on texture synthesis [29], we interpolate the inverse cumulative distribution functions (CDF): $C_t^{-1} = \sum_{i=1}^{k} w_i\, C_i^{-1} / \sum_{i=1}^{k} w_i$, where C_t is the target CDF created by linearly combining nearby CDFs C_i with weights w_i. We use inverse distances in histogram space as weights: $w_i = 1/d(H_v, H_i)^2$ where H_v is the histogram of the current view, H_i is the unedited histogram of the i-th neighboring edit, and $d(\cdot)$ is the EMD function. Once we have the target inverse CDF, we fit a linear model of brightness and contrast change to best match the inverse CDF of the current view C_v^{-1} to the target. That is, we seek α and β so that $\alpha C_v^{-1} + \beta$ is close to C_t^{-1}. We found that a least-squares solution overly emphasizes large differences in the inverse CDFs and does not account for clipping (values above 1 or below 0). We use an iteratively reweighted least-squares algorithm with weights γ_j that are low outisde $[0;1]$ and that decrease the influence of large differences,

$$\gamma_j = \begin{cases} \epsilon & \text{if } \alpha C_v^{-1}(j) + \beta \notin [0;1] \\ \frac{1}{|\alpha C_v^{-1}(j) + \beta - C_t^{-1}(j)|} & \text{otherwise} \end{cases} \tag{1}$$

where $\epsilon = 0.001$. If we ignore the weights outside $[0;1]$, this scheme approximates a L_1 minimization [19]. Figure 3 and the companion video illustrate our approach.

5. Results

The companion video shows a sample editing session with our display-aware editing prototype. The main advantage of our approach is that editing is interactive. In comparison, seamless cloning using Adobe Photoshop can take several minutes for large copied region. Because of its slowness, retouching with a tool such as Photoshop is limited to the most critical points and overall, the image is left untouched, as it has been captured. Our approach addresses this issue and makes it easier to explore creative edits and variations since feedback is instantaneous.

5.1. Complexity Analysis

We analyze the computational complexity of our editing approach by first looking at the cost of fetching the visible data from our data structure and then at the editing algorithms.

Preparing the Visible Data For a $w_{\text{dis}} \times h_{\text{dis}}$ display and $w_{\text{tile}} \times h_{\text{tile}}$ tiles, the number of tiles that we load is less than $(w_{\text{dis}}/w_{\text{tile}} + 1) \times (h_{\text{dis}}/h_{\text{tile}} + 1)$. When we apply geometric transformations to the tiles, these introduce limited deformations and can be taken into account with a small increase of w_{tile} and h_{tile}. Since we have pre-computed the tiles at all $\frac{1}{2^n}$ scales, we load at most four times as many pixels as needed. Last, we may have several input images but we do not load any data for the images outside the current view. Put together, this ensures that we handle an amount of data on the order of $O(k_{\text{dis}}\, \aleph_{\text{dis}})$ where k_{dis} is the number of visible input images and $\aleph_{\text{dis}} = w_{\text{dis}} \times h_{\text{dis}}$ is the resolution of the display. With our scheme, loading the visible image data has a cost linear with respect to the display size. This is important in applications where images are transmitted, e.g., from a photo sharing website to a mobile device.

Editing Operators The per-pixel processes such as the viewing adjustments and the classifier-based selection are in $O(\aleph_{\text{dis}})$ since they do a fixed amount of computation for each pixel. The pyramid-based operators such as texture enhancement runs the same process for each pyramid coefficient. Since a pyramid has $4/3$ times as many pixels as the image, these operators are also linear with respect to the display resolution $\aleph_{\text{dis}}$. The stitching operator processes all the k_{dis} visible images, which introduces a factor k_{dis}. This ensures a $O(k_{\text{dis}}\, \aleph_{\text{dis}})$ complexity, and since loading the data is also linear, our entire pipeline has a linear complexity with respect to the display size.

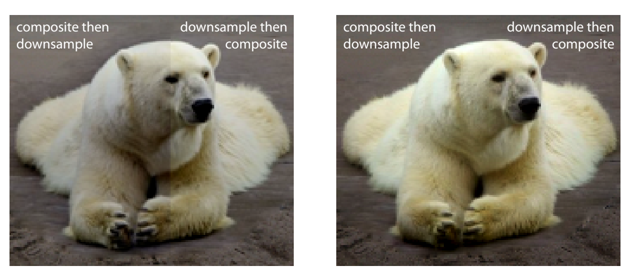
5.2. Accurate Results from Low Resolution Only

We verify that our operators commute with the screen operator discussed in Section 3.1 by comparing their results computed at full resolution rescaled to the screen resolution with the result computed directly from the data at screen resolution. Figure 4 shows that our push-pull compositing produces indistinguishable results in both cases, that is, we can compute the exact result directly at screen resolution without resorting to the full-resolution data. In comparison, the scheme used in Photoshop [20] produces significantly different outputs.

We performed the same test for image stitching using Photoshop and our scheme (§ 3.2). Both produce visually indistinguishable results, however Photoshop is significantly slower because even its optimized solver [5] becomes slow on large images, e.g. a minute or more for several high-resolution images. In comparison, our scheme runs interactively and is grounded on a theoretical study (§ 3.1).

5.3. Running Times

We tested our prototype editing system on a Windows PC equipped with an Intel Xeon 3.0 GHz CPU with 16 GB of system memory and an NVIDIA Quadro FX 5800 GPU with 4 GB of graphics memory. Figure 5 provides the performance result of our system. We measured the average frame rate of the system while applying the global operators



(a) Photoshop (24dB) (b) our method (45dB)

Figure 4. For Photoshop and our approach, we compute a composite and then downsample it (shown on the left halves of the images) and compare the output to the composite computed directly on downsampled data (the right halves). Whereas Photoshop produces different results (a), our method generates visually indistinguishable images (b).

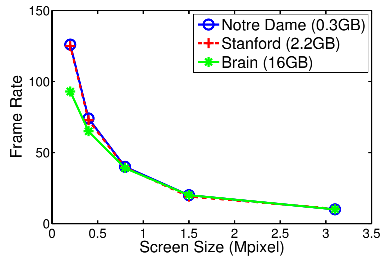


Figure 5. Timings of the global operators running on various screen and input sizes. The performance of our display-aware algorithms depends on the screen size and not on the size of the data.

to the image at arbitrary locations. We gradually change the viewpoint and zoom level during the test to reduce cache memory effect in a realistic setup. Our timings include data transfers so that we measure the time that a user actually perceives when working with our prototype. Note that I/O operations are often excluded from the measures of other methods, e.g. [17].

We tested the operators on five different screen sizes, from 512×384 (0.2 megapixels) to 2048×1536 (3 megapixels), and three different size of input images, from 0.3 to 16 gigapixels. The result shows the benefit of display-aware editing: the frame rate is not affected by the input image size (three plots are almost identical in Figure 5) but is highly correlated with the screen size (frame rates drop as the screen size increases in Figure 5). Note that the 16-gigapixel brain image is much larger than the size of graphics memory we used, but the frame rate is similar to a 0.3-gigapixel image. In addition, the construction of the Gaussian and Laplacian pyramids for a 1024×768 screen resolution took only 11 ms, which enables the execution pyramid-based image operators on-the-fly without using a pre-built global image pyramid. Our on-the-fly image registration runs on a fixed-size grid and is highly parallelizable, and takes 50 to 100 ms in our prototype implementation. The numbers in Figure 5 show that our algorithms are fast and that our data management strategy successfully prevents data starvation.

5.4. Validation of our Interpolation Scheme

We validate our algorithm for propagating viewing parameters on the user study data described in Section 4.1. The data consists of edits from 20 users on 5 views from each of 10 different panoramas (a total of 50 images). Using a leave-one-out strategy for each panorama, we predict one view using the user edits from the 4 other views. We use the Earth Mover's Distance between the histograms of our predicted edit and the user's actual edit to quantify the accuracy of our prediction. On average, the difference is 3.0 with a standard deviation of 1.9. We compared our interpolation scheme to simply interpolating the users' brightness

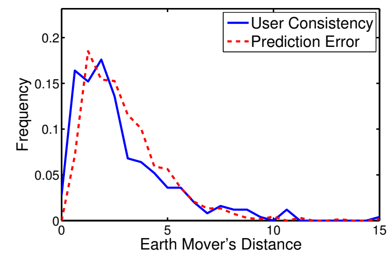


Figure 6. Distribution of the differences between users' edits and our predictions, and between users' edits on repeated images (see text for details). The similarity between these distributions indicates that our edit propagation reproduces users' adjustments.

and contrast adjustments between views (i.e., interpolating the slider positions instead of the histograms). Compared to the users' actual edits, this interpolation scheme produced an average error of 3.7 with a standard deviation of 2.1. A two-sample t-test confirms that our histogram interpolation sheme has a lower error than interpolating the adjustments with a p-value below 10^{-8}. To put these errors into perspective, we conducted an second study in which users edited 20 images comprising 5 images appearing twice and 10 distractors. The image order was randomized such that repeated images were not back to back. We collected 250 repeated measurements and on average, the difference was 2.8 with a standard deviation of 2.3. This result shows that our scheme reproduces users' adjustments within a margin comparable to their own repeatability. Figure 6 illustrates this point.

6. Conclusions and Future Work

Our display-aware image editing framework can effectively handle images that otherwise would be difficult and slow to process. A large part of the benefits of our approach comes from the fact that we process only the visible data. When one needs the whole image at full resolution, for instance to print a poster, we will have to touch every single pixel and the running times are slower. Even in those cases our method remains fast since our editing algorithms are data parallel. In addition, all our algorithms use the same scale-space data structure and apply very similar operations to it, which makes data management and out-of-core processing easier. We envision a workflow in which the user would first edit the image on screen, thereby enjoying the speed of our display-aware approach, and run a final rendering at the end, just before sending the result to an output device such as a printer.

Although we have shown that we can support a variety of tasks with our display-aware approach, there are a few cases that are difficult. Optimization-based techniques require to access every pixel which makes them overly slow on large images. This prevents the use of some algorithms such as error-tolerant and highly discriminative selections [6, 26]. Related to this issue, algorithms akin to histogram equalization manipulate every pixel and become unpractical on large images. A solution is to apply them at lower resolution and to upsample their results [23]. Nonetheless, developing a display-aware version of these algorithms is an interesting avenue for future work. We also imagine that other novel display-aware algorithms will be developed in the future. Ultimately, processing and data storage are getting cheaper, making the need for on-the-fly computation of large images more pressing. In addition, we envision that our framework could be efficiently implemented to edit high-resolution photographs, e.g., from a digital SLR, on commodity mobile devices.

References

[1] GigaPan. www.gigapan.org.
[2] PixOrb. www.peaceriverstudios.com/pixorb.
[3] A. Adams. *The Print: Contact Printing and Enlarging*. Morgan and Lester, 1950.
[4] A. Adams, N. Gelfand, J. Dolson, and M. Levoy. Gaussian kd-trees for fast high-dimensional filtering. *ACM Trans. on Graphics*, 28(3), 2009.
[5] A. Agarwala. Efficient gradient-domain compositing using quadtrees. *ACM Trans. Graph.*, 26(3):94, 2007.
[6] X. An and F. Pellacini. Appprop: All-pairs appearance-space edit propagation. *ACM Trans. on Graphics*, 27(3), 2008.
[7] Anonymous. Interactive histology of large-scale biomedical image stacks, 2010. Submitted to VIS. Available in supplemental material.
[8] S. Bae, S. Paris, and F. Durand. Two-scale tone management for photographic look. *ACM Trans. on Graphics*, 25(3), 2006.
[9] I. Bankman, editor. *Handbook of Medical Imaging: Processing and Analysis Management (Biomedical Engineering)*. Academic Press, 2000.
[10] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman. PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Trans. on Graphics*, , 2009.
[11] D. F. Berman, J. T. Bartell, and D. H. Salesin. Multiresolution painting and compositing. In *Proc. of the ACM SIGGRAPH conference*, 1994.
[12] M. Brown and D. G. Lowe. Automatic panoramic image stitching using invariant features. *International Journal of Computer Vision*, 74(1), 2007.
[13] P. J. Burt. Moment images, polynomial fit filters, and the problem of surface interpolation. In *Proc. of Computer Vision and Pattern Recognition*, 1988.
[14] P. J. Burt and E. H. Adelson. A multiresolution spline with application to image mosaics. *ACM Trans. on Graphics*, 2(4), 1983.
[15] J. Chen, S. Paris, and F. Durand. Real-time edge-aware image processing with the bilateral grid. *ACM Trans. on Graphics*, 26(3), 2007.
[16] A. Criminisi, T. Sharp, C. Rother, and P. Perez. Geodesic image and video editing. *ACM Trans. on Graphics*, 2010.
[17] Z. Farbman, G. Hoffer, Y. Lipman, D. Cohen-Or, and D. Lischinski. Coordinates for instant image cloning. *ACM Trans. Graph.*, 28(3), 2009.
[18] R. Fattal. Edge-avoiding wavelets and their applications. *ACM Trans. on Graphics*, 28(3), 2009.
[19] J. Gentle. *Matrix algebra*, chapter 6.8.1 Solutions that Minimize Other Norms of the Residuals. Springer, 2007.
[20] T. Georgiev. Covariant derivatives and vision. In *Proc. of the European Conference on Computer Vision*, 2006.
[21] K. J. Hayworth, N. Kasthuri, R. Schalek, and J. W. Lichtman. Automating the collection of ultrathin serial sections for large volume TEM reconstructions. In *Microscopy and Microanalysis*, 2006.
[22] M. Kazhdan and H. Hoppe. Streaming multigrid for gradient-domain operations on large images. *ACM Trans. on Graphics*, 27(3), 2008.
[23] J. Kopf, M. F. Cohen, D. Lischinski, and M. Uyttendaele. Joint bilateral upsampling. *ACM Trans. on Graphics*, 26(3), 2007.
[24] J. Kopf, M. Uyttendaele, O. Deussen, and M. F. Cohen. Capturing and viewing gigapixel images. *ACM Trans. on Graphics*, 26(3), 2007.
[25] A. Levin, A. Zomet, S. Peleg, and Y. Weiss. Seamless image stitching in the gradient domain. In *Proc. of the European Conference on Computer Vision*, 2006.
[26] Y. Li, J. Sun, C.-K. Tang, and H.-Y. Shum. Lazy snapping. *ACM Trans. on Graphics*, 2004.
[27] H. Ling and K. Okada. An efficient earth mover's distance algorithm for robust histogram comparison. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 29(5), 2007.
[28] J. Liu, J. Sun, and H.-Y. Shum. Paint selection. *ACM Trans. on Graphics*, 28(3), 2009.
[29] W. Matusik, M. Zwicker, and F. Durand. Texture design using a simplicial complex of morphable textures. *ACM Trans. on Graphics*, 24(3), 2005.
[30] J. McCann and N. S. Pollard. Real-time gradient-domain painting. *ACM Trans. on Graphics*, 27(3), 2008.
[31] A. Oliva and A. Torralba. Modeling the shape of the scene: a holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3), 2001.
[32] S. Paris and F. Durand. A fast approximation of the bilateral filter using a signal processing approach. *International Journal of Computer Vision*, 2009.
[33] P. Pérez, M. Gangnet, and A. Blake. Poisson image editing. *ACM Trans. Graph.*, 22(3), 2003.
[34] S. Pinheiro and L. Velho. A virtual memory system for real-time visualization of multi-resolution 2D objects. *Journal of WSCG*, 2002.
[35] M. A. Shantzis. A model for efficient and flexible image computing. In *Proc. of SIGGRAPH* , 1994.
[36] L. Velho and K. Perlin. B-spline wavelet paint. *Revista de Informatica Teorica e Aplicada*, 2002.
[37] S. Wang and W. Heidrich. The design of an inexpensive very high resolution scan camera system. In *Proc. of Eurographics*, 2004.