



LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Interactive Gradient Editing of Massive Imagery Made Simple: Turning Atlanta into Atlantis

V. Pascucci, B. Summa, G. Scorzelli, M. Jiang, P. Bremer

September 2, 2010

ACM Transaction on Graphics

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

Interactive Gradient Editing of Massive Imagery Made Simple: Turning Atlanta into Atlantis

Valerio Pascucci, Brian Summa, Giorgio Scorzelli*,
SCI Institute, University of Utah

Ming Jiang, Peer-Timo Bremer†
Lawrence Livermore National Laboratory



Figure 1: *The Edinburgh Panorama* 16, 950 × 2, 956 pixels. (Top) Our coarse solution computed at a resolution of 0.7 megapixels; (Middle) The same panorama solved at full resolution with our progressive global solver scaled to approximately 12 megapixel for publication; (Bottom Left) A detail view of a particularly bad seam from the original panorama; (Bottom Middle) The problem area previewed using our adaptive local refinement; (Bottom Right) The problem area solved at full resolution using our global solver in 3.48 minutes.

Abstract

This paper presents a simple framework for progressive processing of high-resolution images with minimal resources. We demonstrate this framework’s effectiveness by implementing an adaptive, multi-resolution solver for gradient-based image processing that, for the first time, is capable of handling gigapixel imagery in real time. With our system, artists can use commodity hardware to interactively edit massive imagery and apply complex operators, such as seamless cloning, panorama stitching, and tone mapping.

In particular, we introduce a progressive Poisson solver that processes images in a purely coarse-to-fine manner, providing near instantaneous global approximations for interactive display (see Figure ??). We also allow for data-driven adaptive refinements to locally emulate the effects of a global solution. These techniques, combined with a fast, cache-friendly data access mechanism, allow the user to interactively explore and edit massive imagery, with the illusion of having a full solution at hand. In particular, we demonstrate the interactive modification of gigapixel panoramas that previously required extensive offline processing. Even with massive satellite images surpassing a hundred gigapixels in size, we enable repeated interactive editing in a dynamically changing environment. Images at these scales are significantly beyond the purview of previous methods yet are processed interactively using our techniques. Finally our system provides a robust and scalable out-

of-core solver that consistently offers high quality solutions while maintaining strict control over system resources.

Keywords: Poisson equation, gradient domain editing, gigapixel images, out-of-core processing, cache-oblivious data layout

1 Introduction

Gigapixel images are increasingly popular due to the availability of high-resolution cameras and inexpensive robots for the automatic capture of large image collections [GigaPan]. These tools simplify the acquisition of large, stitched panoramas, which are becoming easily accessible over the internet. Even larger images, massive in size, are freely distributed, such as aerial satellite photography from the United States Geological Survey (USGS) website. Yet, the full potential of such imagery is only realized by artists and analysts enhancing, manipulating, and/or compositing the original images. However, such editing typically requires significant offline processing and computing resources beyond what can be typically expected.

Poisson Image Processing. A variety of gradient-based methods provide a popular, but computationally expensive, set of techniques for advanced image manipulation. Given a guiding gradient field constructed from one or multiple source images, these techniques attempt to find a closest-fit image using some predetermined distance metric. This basic concept has been adapted for standard image editing [Pérez et al. 2003], as well as more advanced mat-

* {pascucci, bsumma, scrgiorgio} @sci.utah.edu

† {jiang4, bremer5} @llnl.gov

ting [Sun et al. 2004] operations, and high level drag-and-drop functionality [Jia et al. 2006]. Furthermore, gradient-based techniques can tone map high dynamic range images to display favorably on standard monitors [Fattal et al. 2002] or hide the seams in panoramas [Pérez et al. 2003; Levin et al. 2004; Agarwala et al. 2004; Kazhdan and Hoppe 2008]. Other applications include detecting lighting [Horn 1974] or shapes from images [Weiss 2001], removing shadows [Finlayson et al. 2002] or reflections [Agrawal et al. 2005], and gradient domain painting [McCann and Pollard 2008]. Recently, an alternative approach using Mean-Value Coordinates has smoothly interpolated the boundary offset between source images, thereby mimicking Dirichlet boundary conditions [?]. This promising new line of research has yet to show support of Poisson techniques such as tone-mapping, the ability to work well out-of-core, or consistently acceptable results for methods that typically require Neumann boundary conditions.

Poisson Solvers. The solution to a 2D Poisson problem lies at the core of gradient based image processing. Poisson equations have wide utility in many engineering and science applications. Computing their solution efficiently has been the focus of a large body of work and even a cursory review is beyond the scope of this paper. For small images, methods exist to find the direct analytical solution using Fast Fourier transforms [?; ?; ?]. Simchony [?] provides a survey of these methods for computer vision applications. Often the problem is simplified by discretization into a large linear system whose dimension is typically the number of pixels in an image. If this system is small enough to fit into memory, methods exist to find the direct solution and we refer the reader to Dorr [1970] who provides an extensive review on direct methods. Typically, iterative Krylov subspace methods, such as conjugate gradient, are used due to their fast convergence. For much larger systems, memory consumption is the limiting factor and iterative solvers, such as Successive Over-Relaxation (SOR) [Axelsson 1994], become more attractive.

Depending on the application, different levels of accuracy may be required. Sometimes, a coarse approximation is sufficient to achieve the desired result. Bilateral upsampling methods [?] operating on a coarse solution produced good results for applications such as tonemapping. Such methods have not yet been shown to handle applications such as image stitching where the interpolated values are typically not smooth at the seams between images.

When pure upsampling is insufficient, the system must be solved fully. Multigrid methods are often employed to aid the convergence of an iterative solver. Such methods have proven particularly effective by dealing with the large scale trends at coarse resolutions. These techniques include preconditioners [Gortler and Cohen 1995; Szeliski 2008] and multigrid solvers [Brandt 1977; Briggs et al. 2000]. There exist different variants of multigrid algorithms using either adaptive [Berger and Colella 1989; Kazhdan et al. 2006; Bolitho et al. 2007; Agarwala 2007; Ricker 2008] or non-adaptive meshes [Kazhdan 2005; Kazhdan and Hoppe 2008]. As a first step in a complete multigrid system, the mesh is coarsened. The Poisson equation can then be solved in a coarse-to-fine manner. One full iteration, from fine to coarse and back, is typically called a V-cycle. Most recently, a V-cycle was implemented in a streaming fashion for large panoramas [Kazhdan and Hoppe 2008]. However, other systems only implement part of the V-cycle. Kopf et al. [2007] implement only the second half in a pure upsampling procedure, while Bolitho et al. [2007] implement a purely coarse-to-fine solver also called cascadic [Bornemann and Krause 1996]. Lischinski et al. [2006] applied this pure coarse-to-fine approach for interactive tonal adjustment. Our method (for the first time) shows that a cascadic approach has applications well beyond the adjustment of tonal values and can be used for a wide variety of gradient based image processing techniques. We also extend such techniques to al-

low the interactive editing and processing of gigapixel images. As discussed in Section 2.2, the solver propagates sufficient information from coarse-to-fine, allowing us to achieve local solutions at interactive rates that are virtually indistinguishable from the full-resolution solution.

Out-of-Core Methods. Toledo [Toledo 1999] presents a survey of general out-of-core algorithms for linear systems. The majority of algorithms surveyed assume that at least the solution vectors can be kept in main memory, which is not the case for large images. For out-of-core processing of large images, the streaming multigrid method of Kazhdan and Hoppe [2008] has so far provided the only solution. However, processing a three gigapixel image using this technique still takes well over an hour which does not support an interactive trial-and-error artistic process. Many algorithms such as tone mapping require careful parameter tuning to achieve good results. Thus, waiting hours to examine the effects of a single parameter change is not feasible in this context.

An additional disadvantage of traditional out-of-core methods is their tendency to achieve a low memory footprint at the cost of significantly proliferating the disk storage requirements. For example, the multigrid method [Kazhdan and Hoppe 2008] requires auxiliary storage an order of magnitude greater than the input size, almost half of which is due to gradient computation. In contrast our approach completely avoids such data proliferation, thereby allowing the processing of data which already pushes the boundaries of available storage.

The multigrid method [Kazhdan and Hoppe 2008] is also limited by main memory usage since it is proportional to the number of iterations of the solver. This can cause the method to not achieve acceptable results for images that may require a large number of iterations, as shown in Section 4. Our method's memory usage is independent of the number of iterations of the Poisson solver and, therefore, scales gracefully in these cases.

Out-of-Core Data Access. Given an image, it is well known that the standard row-major order exhibits good locality in only one dimension and is therefore ill-suited for an unconstrained out-of-core storage scheme [Vitter 2001]. Previous out-of-core Poisson methods [Kazhdan and Hoppe 2008] have been noted to be severely limited by this constraint. Instead, indexing based on various space-filling curves [Sagan 1994] has been proposed in different applications [Niedermeier et al. 1997; Griebel and Zumbusch 1999; Balmelli et al. 1999; Lawder and King 2000] to exploit their inherent geometric locality. Of particular interest is the Z-order (also called Lebesgue-order) [Balmelli et al. 1999; Pascucci and Frank 2002] since it allows an especially simple conversion to and from standard row-major indices. While Z-order exhibits good locality in all dimensions, it does so only at a fixed resolution and does not support hierarchical access. Instead, our system has the ability to use the hierarchical variant, called HZ-order, proposed by Pascucci and Frank [2002].

Contributions. We introduce a simple and light-weight framework that provides the user with the illusion of a full Poisson system solve at interactive frame rates for image editing. This framework also allows for the computation of a full solution with a simple approach, rivaling the run time of the current best out-of-core technique [Kazhdan and Hoppe 2008] while producing equal or higher quality results on images that require a large number of iterations. Our system is flexible enough to handle different hierarchical image formats such as tiling for higher quality images or HZ-order for greater I/O speed (see Section ??). In particular, through exploiting a new implicit kd-tree hierarchy for HZ-order, the framework needs only to access and solve visible pixels (see Appendix 5). This allows an artist to interactively apply gradient-based techniques to

images gigapixels in size. Our framework is straightforward and requires neither complicated spatial indexing nor advanced caching schemes. In particular our contributions are:

- A coarse-to-fine progressive Poisson solver running at interactive frame rates, extended to a wide variety of gradient domain tasks, with the ability to scale to gigapixel images. This cascadic solver entirely avoids the coarsening stage of the V-cycle yet produces high quality results;
- A method to locally refine solutions having time and space requirements that are linearly dependent on the screen resolution rather than the resolution of the input image;
- A full out-of-core solver that maintains strict control over system resources, rivals the run-times for the best known method and consistently achieves quality results where previous methods may not converge well in practice; and
- A light-weight streaming framework that provides adaptive multi-resolution access to out-of-core images in a cache coherent manner, without using intricate indexing data structures or pre-caching schemes.

Progressive, interruptible computations and adaptive refinements allow our system to maintain interactive rates under constant change of input parameters even while processing the highest resolution of the image data (see Figure ??). As such, this approach achieves a new level of scalability compared to previous techniques based on streaming [Kazhdan and Hoppe 2008] or quad-tree [Agarwala 2007] computations, which allowed for highly efficient but offline computations. In the accompanying video, we demonstrate the practical use of our method for editing massive imagery in an interactive environment.

2 Progressive Poisson Solver

This section briefly reviews the Poisson system at the foundation of gradient domain image processing. Subsequently, we discuss a progressive framework for solving very large Poisson systems in massive image editing. This technique allows for a simple implementation, yet is highly scalable, and performs well even with limited storage and processing resources.

2.1 Gradient Domain Image Processing

Gradient domain image processing encompasses a family of techniques that manipulate an image based on the value of a gradient field rather than operating directly on the pixel values. Seamless cloning, panorama stitching, and high dynamic range tone mapping are all techniques that belong to this class. Given a gradient field $\vec{G}(x, y)$, defined over a domain $\Omega \subset \mathbb{R}^2$, we seek to find an image $P(x, y)$ such that its gradient ∇P fits $\vec{G}(x, y)$.

In order to minimize $\|\nabla P - \vec{G}\|$ in a least squares sense, one has to solve the following optimization problem:

$$\min_P \iint_{\Omega} \|\nabla P - \vec{G}\|^2 \quad (1)$$

It is well known that minimizing equation (1) is equivalent to solving the Poisson equation $\Delta P = \text{div } \vec{G}(x, y)$ where Δ denotes the Laplace operator $\Delta P = \frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2}$ and $\text{div } \vec{G}(x, y)$ denotes the divergence of \vec{G} .

To adapt the equations shown above to discrete images, we apply a standard finite difference approach which approximates the Lapla-

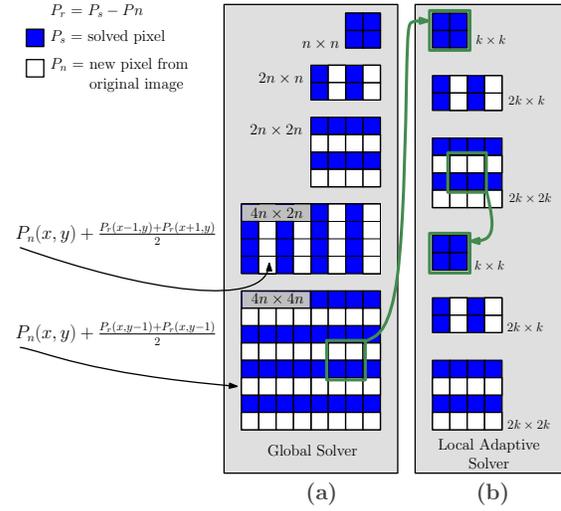


Figure 2: Our adaptive refinement scheme using simple difference averaging. (a) Global progressive up-sampling of the edited image computed by a background process. (b) View-dependent local refinement based on a $2k \times 2k$ window. In both cases we speedup the SOR solver with an initial solution obtained by smooth refinement of the solution.

cian as:

$$\Delta P(x, y) = P(x+1, y) + P(x-1, y) + P(x, y+1) + P(x, y-1) - 4P(x, y) \quad (2)$$

and the divergence of $\vec{G}(x, y) = (G^x(x, y), G^y(x, y))$ as:

$$\text{div } \vec{G}(x, y) = G^x(x, y) - G^x(x-1, y) + G^y(x, y) - G^y(x, y-1).$$

The differential form $\Delta P = \text{div } \vec{G}(x, y)$ can therefore be discretized into the following sparse linear system:

$$L\mathbf{p} = \mathbf{b}. \quad (3)$$

Each row of the matrix L stores the weights of the standard five point Laplacian stencil given by (2), \mathbf{p} is the vector of pixel colors, and \mathbf{b} encodes the guiding gradient field, as well as the boundary constraints. The choice of guiding gradient field $\vec{G}(x, y)$ and boundary conditions for the system determines which image processing technique is applied. In the case of seamless cloning, it is necessary to use Dirichlet boundary conditions, set to be the color values of the background image at the boundaries, and the guiding gradient to be the gradient of the source image (see [Pérez et al. 2003] for a detailed description). For tone mapping and image stitching, Neumann boundary condition are used. The guiding gradient field for image stitching is the composited gradient field of the original images. The unwanted gradient between images is commonly set to zero or averaged across the stitch. The guiding gradient for tone mapping is adjusted from the original pixel values to compress the HDR range (see [Fattal et al. 2002] for more detail). Methods such as gradient domain painting [McCann and Pollard 2008] allow the guiding gradient to be user defined.

2.2 Progressive Framework

For an image P of $n \times n$ pixels, the Laplace system (3) has n^2 independent variables, one per pixel. Computing the entire solution is therefore expensive both in terms of space and time. For

large images, the space requirements easily exceed the main memory available on most computers. Moreover, the long computation times make any interactive application unfeasible.

Acceleration methods try to address either or both of these issues. The recent adaptive formulation by Agarwala [Agarwala 2007] has been particularly insightful. By exploiting the smoothness of the solution, this method was the first to reduce both the cost of the computation and its memory requirements. The approach by Kazhdan and Hoppe [2008] demonstrates how a streaming approach can achieve high performance by optimizing the memory access patterns.

We extend these acceleration techniques and show how to achieve high quality local solutions, without the need for solving the entire system. Moreover, we show that coarse approximations are of acceptable visual quality without the cost of a typical coarsening stage used in the V-cycle. These new features, coupled with a simple multi-resolution framework, enable a data-driven interactive environment that exploits the fact that interactive editing sessions are always limited by screen resolution. At any given time, a user only sees either a low resolution view of the entire image or a high resolution view of a small area. We take advantage of this practical restriction and solve the Poisson equation only for the visible pixels. This provides performance advantages for interactive sessions, as well as tight control over the memory usage. For example, even the simple step of computing the gradient of the full resolution image can be problematic due to its significant processing time and storage requirement. In our approach, we avoid this problem by estimating gradients on-the-fly using only the available pixels.

Overall, our interactive system is based on a simple two tier approach:

- A global progressive solver provides a near instant coarse approximation of the full solution. This approximation can be refined up to a desired solution by a lightweight process, often running in the background and possibly out-of-core. Any time the user changes input parameters, this process is restarted.
- A local progressive solver provides a quick solution for the visible pixels. This process is driven by the interactive viewer and uses as a coarse approximation the best solution available from the global solver.

These components can be coupled with different multi-resolution hierarchies as discussed in the next section.

Initial Solution. At launch, the system computes a coarse image for the initial view. A fast 2D direct method using cosine and Fast Fourier transforms by Agrawal [?: ?] is used for this initial solve for techniques that require Neumann boundaries (stitching, HDR compression). For methods that require Dirichlet boundaries (seamless cloning) we use an iterative method such as SOR. To provide the user with a meaningful preview, we use an initial coarse resolution of one to two megapixel depending on the physical display. We have found, in practice, that the Fast Fourier solver usually gives us this approximation in under 2 seconds. This initial solution is at the core of the progressive refinement defined in the next paragraph.

Progressive Refinement. The goal of progressive refinement is to increase the resolution of our solution either locally or globally. This requires injecting color transport information from coarser to finer resolutions. In doing so, we exploit the fact that the solution, away from the seams, tends to be smooth [Agarwala 2007] and up-sampling the coarse solution gives high quality results in large areas of the image. To improve the solution and resolve the problems at the seams, we run an iterative method, estimating new gradients from the original pixel data of the finer resolution and using the up-sampled values as the initial solution estimate. The finer res-

olution gradient field allows the iterative solver to reconstruct the detailed features of the original image. For the iterative method we allow the use of either conjugate gradient (for faster convergence) or SOR (for minimal memory overhead). The iterative solver is assumed to have converged when the L_2 norm of the residual between iterations is less than 1.0×10^{-3} . In practice there is no perceptible difference between iterations after this condition is met.

Figure 2 shows the refinement process where we assume for simplicity that each resolution doubles each dimension separately and our data is a subsampled hierarchy. In this case, computing each finer resolution is equivalent to adding new rows (or columns) to the coarse resolution. Therefore, we know that each new pixel added has two neighbors from the coarse solution. We can take the average difference from these two neighbors and apply it to the original RGB value of the pixel from the new resolution (see Figure 2 (a)). Since the image is subsampled, the average difference and application to the new pixel is trivial.

In a tiled hierarchy one would need to double both dimensions at the same time, requiring a trivial adjustment to the interpolation. Each new resolution is simply treated as new data and the offset is based on the solution from the previous resolution and the transform between levels.

Local Preview. Combining the coarse, global solve with a progressively refined local preview is all that is necessary for our interactive system. For data requests at resolutions equal or less than our coarse solution, we simply display the available data. As the user zooms into an area, the image is progressively refined in a local region. Since the resolution increase is directly coupled with the decrease in the extent of the local view, the number of pixels that must be processed remains constant (see Figure 2 (b)). This results in a logarithmic run-time complexity and constant storage requirement, which allows our system to gracefully scale to images orders of magnitude larger than previously possible.

Progressive Full Solution. The progressive refinement can be applied globally to compute a full solution. Since, the method requires a very small overhead, it can easily be run as background process during the interactive preview. When a new resolution has been solved, the interactive preview uses the solution as a new coarse approximation, thereby saving computation during the local adaptive phase. Like other in-core methods, this progressive global solver is limited by available system memory. To address this issue, the global solver has the ability to switch modes to a moving-window out-of-core progressive solver.

Out-of-core Solver. The out-of-core solver maintains strict control over memory usage by sweeping the data with a sliding window. The window traverses the finest desired resolution, which can be several levels in the hierarchy from the current available solution. If the jump in resolution is too high, the process can be repeated several times. Within each window, the coarse solution is up-sampled and the new resolution image is solved using the gradients from the desired resolution. Since the window lives at the desired resolution, we never need to expand memory beyond the size of the window. Furthermore, windows are arranged such that they overlap with the previously solved data in both dimensions to produce a smooth solution. The overlap causes the solver to load and compute some of the data multiple times. This overlap has an inherent overhead when compared to an idealized in-core solver. For instance given a $1/x$ overlap, the 4 corners, each $1/x \times 1/x$ in size, are executed 4 times. The 4 strips on the edge of the window, not including the corners, each $1/x \times (1 - 2/x)$ in size are executed 2 times. All other pixels, size $(1 - 2/x) \times (1 - 2/x)$, are executed once. Therefore, the overhead computation for this $1/x$ overlap is given by: $4/x(1 + 1/x)$. Moreover, the I/O overhead can be reduced to $1/x$, since we can

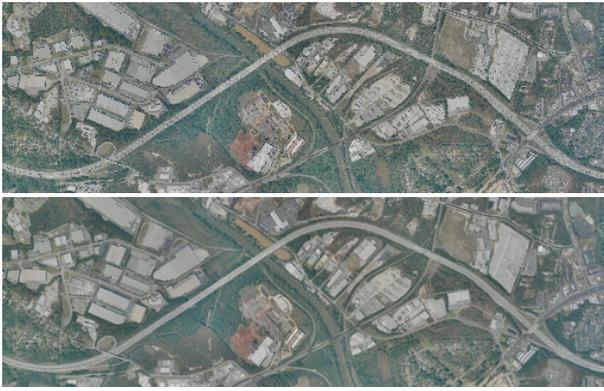


Figure 3: *Subsampled and tiled hierarchies. Top: A subsampled hierarchy. As expected, subsampling has the tendency to produce high-frequency aliasing. Though details such as the cars on the highway and in the parking lots are preserved. Bottom: A tiled hierarchy. This produces a more visually pleasing image at all resolutions but at the cost of potentially losing information. The cars are now completely smoothed away.*

retain pixel data from the previous window in the primary traversal direction. In principle, a larger overlap between windows results in higher quality solutions, though in practice we have found that for a 1024×1024 window a $1/32$ overlap is sufficient for good results (see Figures ?? and 3). This overlap requires only a 12.8% compute overhead and a 3.1% I/O overhead. A larger window can be used to reduce the percentage overlap while achieving the same quality results. For instance, by doubling the window size in both dimensions, a 2048×2048 window can be computed with a $1/64$ overlay, only incurring a 6.3% compute overhead and a 1.5% I/O overhead. Figure ?? shows that, compared to the exact analytical solution, our method produces even higher quality results than the best known method [Kazhdan and Hoppe 2008] for equivalent run times.

3 Data Access

Our progressive solver can operate well on multiple hierarchical schemes. Tiled hierarchies are often used to produce smoother, antialiased images, though high contrast areas in the original image may be lost in the smoothing. As Figure ?? bottom shows, the tiled image is visually pleasing, but details such as the cars on the highway are lost. This visual smoothness can also come at the cost of significant preprocessing, reduced flexibility when dealing with missing data, and increased I/O when traversing the data. The costs can be especially significant for massive data if one has to process it with very limited resources. The least costly image hierarchy can be computed by subsampling. Subsampling is simple and lightweight, but is prone to high frequency aliasing. It does, though, retain higher contrast at the coarse resolution. Figure ?? top shows how the subsampled hierarchy has aliasing artifacts, but also retains enough contrast to see the cars on the highway. This contrast may be beneficial for some applications, such as an analyst studying satellite imagery.

To show the flexibility of our interactive system, we support both a filtered tiled hierarchy and a subsampled hierarchy (see Figure ??). For a tiled scheme, we compute the image hierarchy using a Gaussian kernel to produce a smooth, antialiased image (Figure ?? right column). With a minor variation to the underlying I/O layer, our system also supports a faster, subsampled Hierarchical Z-order as proposed by Pascucci and Frank [2002] (Figure ?? left column). For an overview of the HZ data format, see Appendix ??.

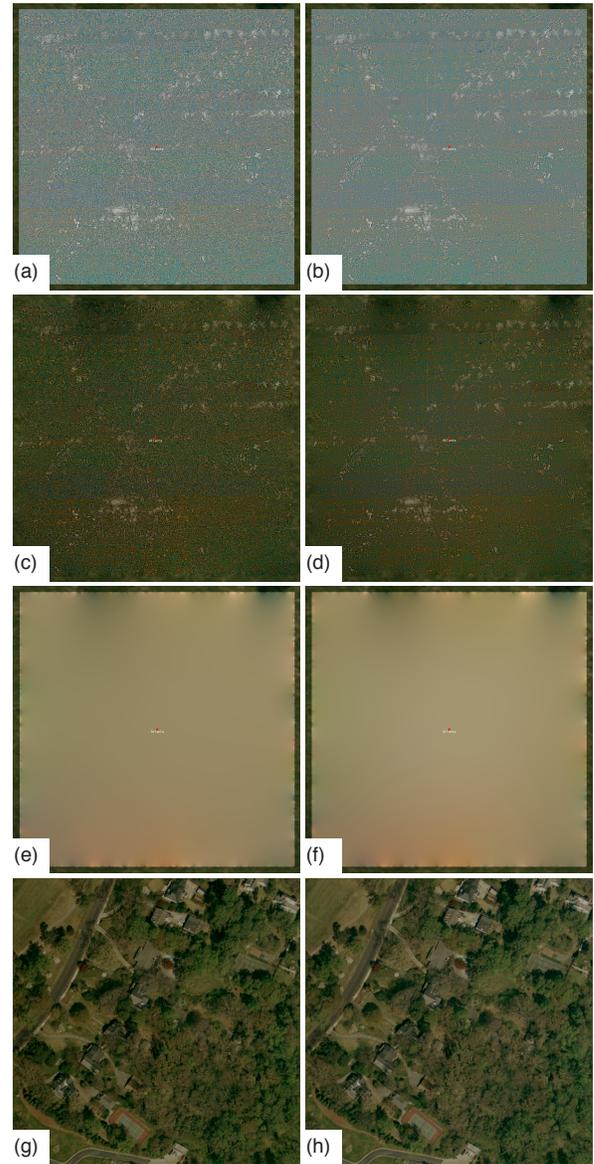


Figure 4: *Our progressive framework using subsampled and tiled hierarchies. (a) A composite satellite image of Atlanta, over 100-gigapixels at full resolution, overlaid on Blue Marble background subsampled; (b) a tiled version of the same satellite image; (c) the seamless cloning solution using subsampling; (d) the same solution computed using a tiled hierarchy; (e) the solution offset computed using subsampling (f) the solution computed using tiles; (g) a full resolution portion computed using subsampling; (h) the same portion using tiling. Note that even though there is a slight difference in the computed solution, both the tiled and the subsampled hierarchies produce a seamless stitch with our framework.*

achieve the level of scalability necessary in the current system, we further simplify the HZ data access scheme. We use a lightweight recursive algorithm that avoids repeated index computations, provides progressive and adaptive access, guarantees cache coherency and minimizes the number of I/O operations without using any explicit caching mechanisms. In particular, computing the HZ index with this new algorithm attains a 30x speedup compared to the previous work. For example, to compute the indices for a 0.8 gigapixel image the new algorithm requires 4.7 seconds where the previous method would take 144.1 seconds. Moreover, since the traversal

follows the HZ storage order exactly for any query window, we guarantee that each file is accessed only once without need of holding any block of data in cache. For details on our new recursive algorithm see Appendix ???. This approach makes the system intrinsically cache friendly for any realistic caching architecture and, therefore, very flexible in exploiting modern hardware. Conversion into HZ-order requires no additional storage. On the other hand, for tiled hierarchies a 1/3 data increase is common. Due to our new data access scheme, conversion to HZ-order is straightforward and inexpensive. For our test data, we have found that there is only a 27% overhead due to the conversion compared to just copying the raw data. In essence, the conversion is strictly a reordering of the data and requires no operations on the pixel data. This conversion will outperform even the most simple tiled hierarchies which require some manipulation of the pixel data.

Each resolution in the HZ-hierarchy is in plain Z-order, which allows for fast, cache coherent access of sub-regions of the image. HZ is not tied to a specific data traversal order, such as the row-major imposed by traditional file formats, as previously observed in [Kazhdan and Hoppe 2008]. In fact, HZ maintains a high degree of cache coherency even during adaptive local traversals. In the accompanying video, we show how the locality of our data access provides graceful performance degradation even in extreme conditions. In particular, we demonstrate accessing a data set, of roughly a terabyte in size, by simply mounting a remote file system over an encrypted VPN channel via a wireless connection. Even in normal running conditions, we have found that the I/O overhead caused by using a tiled hierarchy increased the running time by 39%-67%. These numbers reflect the theoretical bound of 1/3 overhead, made worse by the inability to constrain real queries to perfect alignment with the boundaries of a quadtree. The effect of this overhead is detrimental to the scalability of the system under more difficult running conditions such as the one mentioned above. Moreover, HZ easily handles partially converted data, as we show in one portion of the accompanying video for the editing of the Salt Lake City panorama. In a tiled scheme, the entire hierarchy may need to be recomputed as new data is added.

4 Results

We demonstrate the scalability and interactivity of our approach on several applications, using a number of images ranging from megapixels to hundreds of gigapixels in size. To further illustrate the responsiveness of our system, the accompanying video shows screen captures of live demonstrations. To highlight particular details and validate the approach, the figures in this section show previews and close-ups of our interactive system, alongside the results of our full out-of-core progressive solver. We also provide running times of our full out-of-core solver compared with the best current method, streaming multi-grid [Kazhdan and Hoppe 2008], which we have verified to use the same gradient information. All timings and demos were performed on a 64-bit 2.67 GHz Intel Quad Core desktop, with 8 GB of memory. All streaming multi-grid timings were computed from code provided by the authors and include the timing for the gradient preprocess along with the timing to produce a solution.

Our simple framework provides the illusion of a fully solved Poisson system at interactive frame rates and under continuous parameter changes with only a simple GL texture for display and no special hardware acceleration. Therefore, our code is platform independent. Our simple progressive out-of-core solver produces robust solutions with run times that rival [Kazhdan and Hoppe 2008]. Unlike the previous method, our out-of-core solver does not use hardware acceleration and did not undergo high code optimization to achieve the following runtimes. The solver is also sequential and uses no

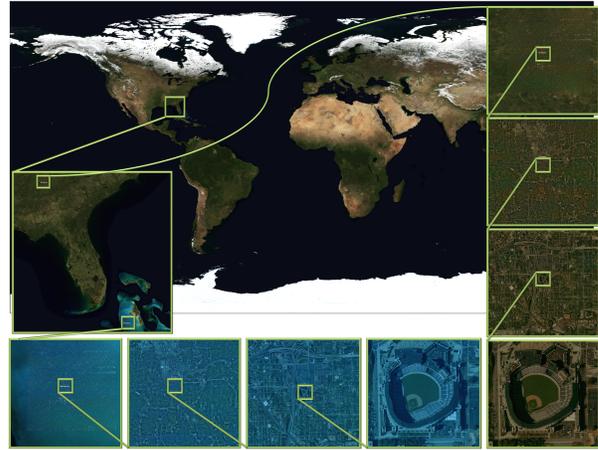


Figure 5: Satellite imagery collection with a background given by a 3.7-gigapixel image from NASA’s Blue Marble Collection. The progressive Poisson solver allows the application of the seamless cloning method to two copies of the city of Atlanta, each of 116-gigapixels. An artist can interactively place a copy of Atlanta under shallow water and recreate the lost city of Atlantis.

threading to accelerate the computation. If further optimization of the run-times is desired, there is nothing in our system to prevent the addition of these acceleration techniques.

Different from other out-of-core methods, we do not rely on large external memory data structures and we do not need to pre-compute gradients for the entire image. For the Salt Lake City panorama, for example, the streaming multi-grid method [Kazhdan and Hoppe 2008] creates 75.2 GB of auxiliary information for a 7.9 GB input image. While disk space is generally assumed to be plentiful, such an explosion in disk space is unsustainable for images hundreds of gigapixel in size. The collection of satellite imagery we use in our video is more than one terabyte in size and would, therefore, require more than 9.5 terabytes of temporary storage.

Edinburgh: 25 images, resolution 16,950 × 2,956, 50-megapixel. At launch, our system performs a seamless stitch Poisson solve of a global 0.7-megapixel image in 1.26 seconds using our direct analytical solver (see Figure ?? top). From this point on, the system can pan and zoom interactively as if the full-solution were already available. Our local adaptive refinement gives a solution that is visually equivalent to a solution to the entire system (see Figure ?? bottom row). In the accompanying video, we demonstrate interactive editing and solving of the Poisson system, after the repeated user selected replacement of pixels of a particular color. We also perform a seamless clone of a 2000 × 1600 airplane on Edinburgh’s cloudy sky. The plane is animated along a linear path across the panorama. As evident in the video, our framework shows the entire sequence in real-time. We also demonstrate similar interactive editing with the Redrock panorama: 9 images, 19,588 × 4,457; 87-megapixels. Given this initial coarse solution, our method can produce a full solution of Edinburgh (see Figure ?? middle) in 3.48 minutes. The streaming multigrid method requires 3.52 minutes. Figure ?? (left) shows the convergence and error for our method and streaming multigrid when compared to the ideal direct solution.

Salt Lake City: 611 images, 126,826 × 29,633, 3.27-gigapixel. A significantly larger example is provided by a panorama captured with a simple camera mounted on a GigaPan robot [GigaPan]. To maximize individual image quality the pictures were taken with automatic exposure times, which inherently increases the color differences between images that need to be corrected by the Poisson

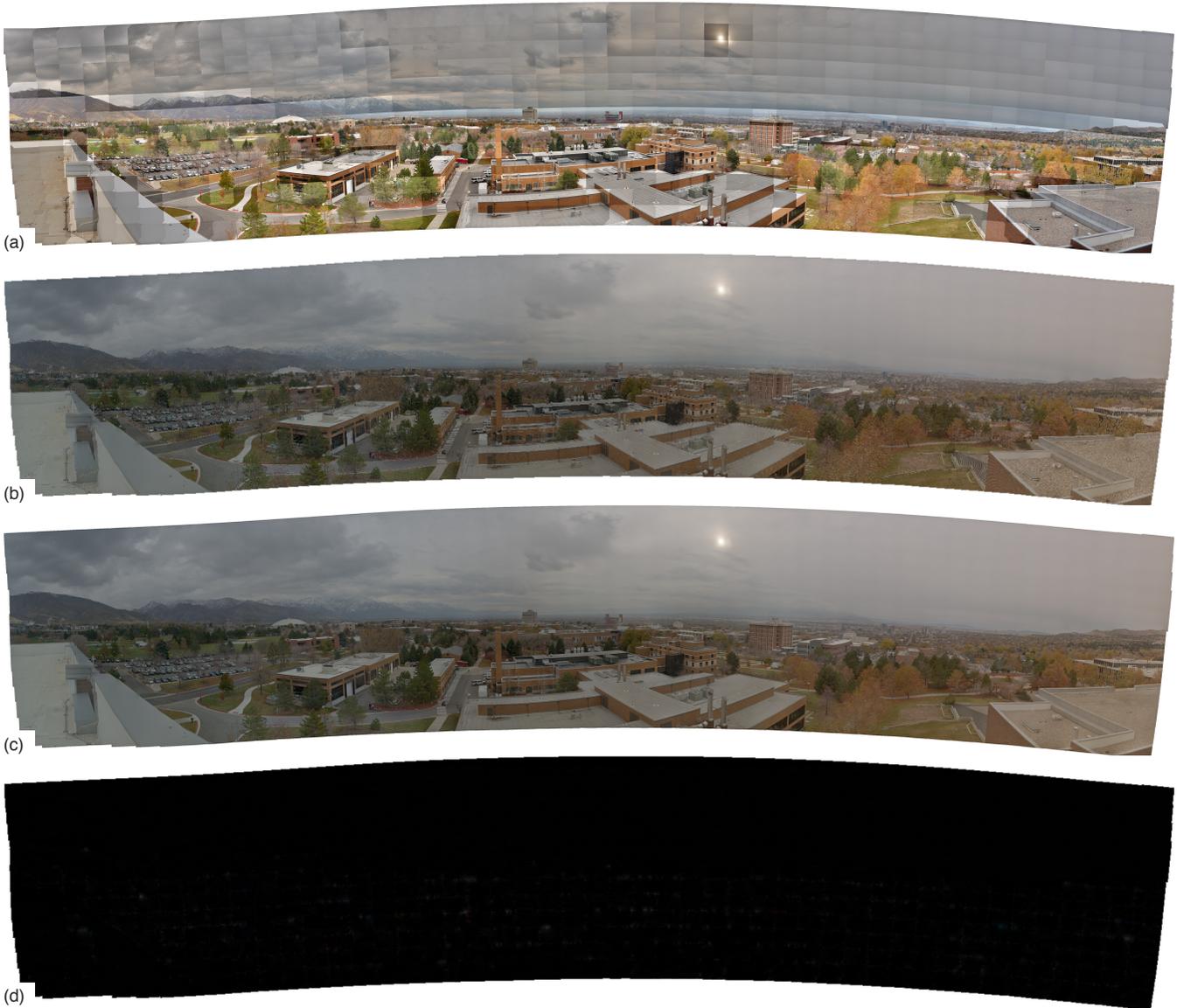


Figure 6: Panorama of Salt Lake City of 3.27-gigapixel, obtained by stitching 611 images. (a) Mosaic of the original images. (b) Our solution computed at 0.9-megapixel resolution. (c) The full solution provided by our global solver. (d) The difference image between our preview and the full solution at the preview resolution. Both (a) and (c) have been scaled for publication to approximately 12.9-megapixels.

solver. An initial coarse preview of 0.87 megapixel is computed by our direct analytical solver in 2.07 seconds. Figure 3 shows the original set of images (a), the panorama that our systems stitches in real time (b), the global solution provided by our out-of-core solver (c), and the difference image between the interactive preview and the final solution at the coarse resolution (d). There are slight deviations at some of the more challenging seams, but overall there is negligible visible difference. Our local adaptive preview mimics well the global solution, as shown in Figure 4. To test the accuracy of the methods, we have run a full analytical Poisson solver on a 485 megapixel subset of the panorama on a HPC-computer. Figures ?? (a) and (b) show how close our out-of-core solution comes to the exact analytical solution. Figure ?? (c) shows that the multigrid method has yet to converge to an acceptable solution given an equivalent amount of running time. All solutions were computed using the map file in Figure ?? (g) where the red channel denotes the image that contributes the color for the pixel in the final panorama and white denotes the panorama boundary. Finally, all methods allow a harmonic fill of this boundary which is shown in

Figure ?? (d), (e), and (f). Figure ?? (right) shows the convergence and error for our method and streaming multigrid when compared to the ideal direct solution. In particular, our out-of-core solver can compute the final panorama in 4.16 hours. At 24 iterations, the multigrid method has yet to converge to a proper solution after running for 4.4 hours. Unfortunately any additional iterations cause the multigrid code to fail, since the number of iterations linearly increases the memory usage of the method.

Sierpinski Sponge: resolution $128k \times 128k$, 16-gigapixel. We have tested the tone mapping application on a synthetic high dynamic range image generated with [?]. In this image we use a partially refined model of a Sierpinski Sponge to create high variations in level-of-detail. Such details can be completely hidden in the dark areas under projected shadows. We follow the approach introduced by Fattal [Fattal et al. 2002] to reconstruct the information hidden in the dark regions. To validate the approach, we've run a typical HDR test image, the Belgium House, progressively refined from a 16×12 coarse solution. Even with such a coarse initial solution,

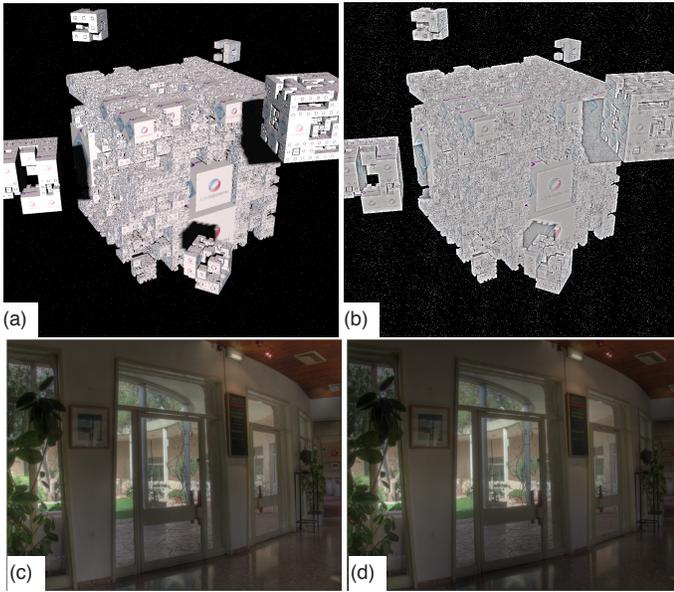


Figure 7: Application of our method to HDR image compression (a) Original synthetic HDR image of an adaptively refined Sierpinski sponge generated with Povray. (b) Tone mapped image with recovery of detailed information previously hidden in the shadows (c) the Belgium House image solved using our coarse-to-fine method with an initial 16×12 coarse solution ($\alpha = 0.01$, $\beta = 0.7$, compression coefficient=0.5) (d) the direct analytical solution

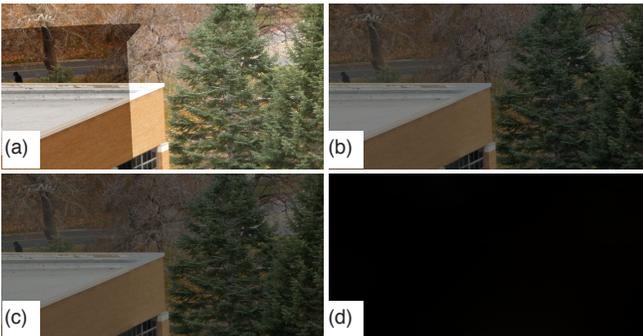


Figure 8: A comparison of our adaptive local preview on a portion of the Salt Lake City panorama $1/2$ of the full resolution; (a) the original Mosaic (b) our adaptive preview (c) the full solution from our global solver. (d) the difference image between the adaptive preview and the full solution

we achieve results very close to the exact solution (see Figure ?? (c) and (d)). Figure ?? shows the original sponge model (a) and the processed version (b), where all the details under the shadows have been recovered.

Satellite Imagery: Blue Marble background, 3.7-gigapixel; Atlanta and other cities, over 100-gigapixel. To demonstrate the scalability of our system, we have run the seamless cloning algorithm for entire cities over a variety of realistic backgrounds from NASA’s Blue Marble Collection [?] (see Figure ??). In the accompanying video, we show how a user can take advantage of these capabilities to achieve artistic effects and create virtual worlds from real data. We also create a dynamic environment by animating the background world map over 12 months and concurrently use the Poisson solver to show how the appearance of a city would change across the seasons.

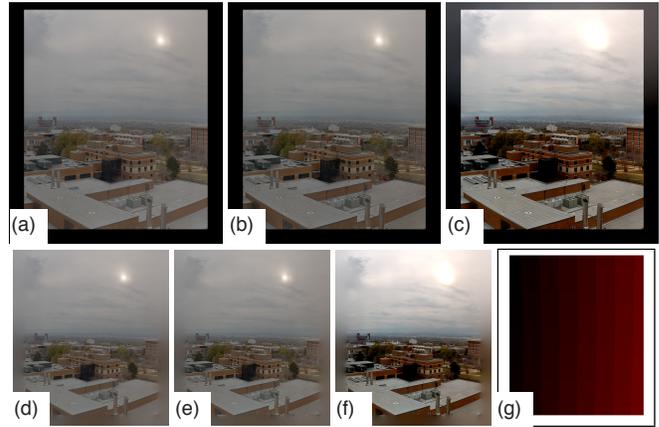


Figure 9: A comparison of our system with the best known out of core method [Kazhdan and Hoppe 2008] and a full analytical solution on a portion of the Salt Lake City panorama, 21201×24001 pixels, 485-megapixel (a) the full analytical solution; (b) our solution computed in 28.1 minutes; (c) solution from [Kazhdan and Hoppe 2008] computed in 24.9 minutes; (d) the analytical solution where the solver is allowed to harmonically fill the boundary; (e) our solution with harmonic fill; (f) solution from [Kazhdan and Hoppe 2008] with harmonic fill; (g) the map image used by all solvers to construct the panorama where the red color indicates the image that provides the pixel color and white denotes the panorama boundary.

5 Conclusions

This paper describes a new light-weight system that allows, for the first time, interactive performance for gradient domain image processing on massive images surpassing 100-gigapixels in size. We achieve this result by combining (i) a purely coarse-to-fine progressive Poisson solver, (ii) an adaptively refined mechanism integrated with the Poisson system, (iii) an out-of-core solver to provide a full solution, and (iv) a fast cache coherent data access method for large images.

At any time, the system provides a user with the illusion of a fully converged solution, while allowing interactive changes in resolution and parameters. The new framework represents a novel paradigm for interacting with extreme resolution imagery, based on computing screen resolution sized previews in real-time and using out-of-core computation without the need of expanding the input size by orders of magnitude.

One can envision expanding this framework with other image processing tools to allow comprehensive editing of massive images on regular desktop computers. Such expansion seems now to be an attainable goal since the Poisson solver presented in this paper is one of the most sophisticated and computationally intensive image editing techniques.

References

- AGARWALA, A., DONTCHEVA, M., AGRAWALA, M., DRUCKER, S., COLBURN, A., CURLESS, B., SALESIN, D., AND COHEN, M. 2004. Interactive digital photomontage. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, ACM, New York, NY, USA, 294–302.
- AGARWALA, A. 2007. Efficient gradient-domain compositing using quadtrees. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, ACM, New York, NY, USA, 94.
- AGRAWAL, A. K., CHELLAPPA, R., AND RASKAR, R. 2005. An algebraic approach to surface reconstruction from gradient fields. In *ICCV*, 1: 174–181.

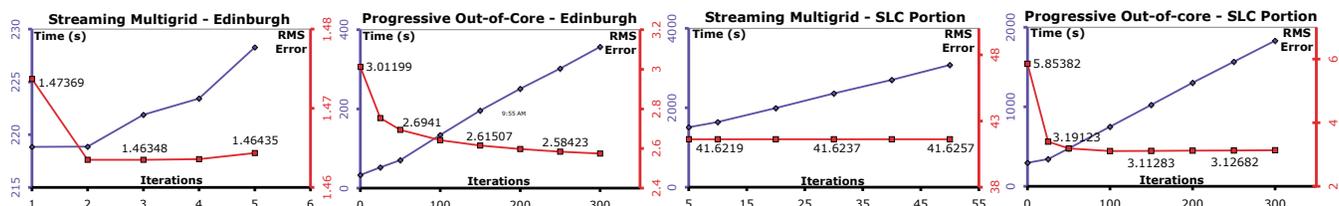


Figure 10: The RMS error when compared to the ideal analytical solution as we increase iterations for both methods. Streaming multigrid has better convergence and less error for the Edinburgh example (left), though our method remains stable for the larger Salt Lake City panorama (right). Notice that every plot has been scaled independently to best illustrate the convergency trends of each method.

AGRAWAL, A., RASKAR, R., NAYAR, S. K., AND LI, Y. 2005. Removing photography artifacts using gradient projection and flash-exposure sampling. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, ACM, New York, NY, USA, 828–835.

AGRAWAL, A. K., RASKAR, R., AND CHELLAPPA, R. 2006. What is the range of surface reconstructions from a gradient field? In *ECCV*, I: 578–591.

AXELSSON, O. 1994. *Iterative Solution Methods*. Cambridge University Press, New York, NY.

BALMELLI, L., KOVACEVIC, J., AND VETTERLI, M. 1999. Quadrees for embedded surface visualization: Constraints and efficient data structures. In *Proc. of IEEE International Conference on Image Processing (ICIP)*, 487–491.

BERGER, M. J., AND COLELLA, P. 1989. Local adaptive mesh refinement for shock hydrodynamics. *Journal Computational Physics* 82, 64–84.

BOLITHO, M., KAZHDAN, M., BURNS, R., AND HOPPE, H. 2007. Multilevel streaming for out-of-core surface reconstruction. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 69–78.

BORNEMANN, F. A., AND KRAUSE, R. 1996. Classical and cascadic multigrid - a methodological comparison. In *Proceedings of the 9th International Conference on Domain Decomposition Methods*, Domain Decomposition Press, 64–71.

BRANDT, A. 1977. Multi-level adaptive solutions to boundary-value problems. *Mathematics of Computation* 31, 138, 333–390.

BRIGGS, W. L., HENSON, V. E., AND MCCORMICK, S. F. 2000. *A Multigrid Tutorial*, 2nd ed. SIAM.

DORR, F. W. 1970. The direct solution of the discrete poisson equation on a rectangle. *SIAM Review* 12, 2 (April), 248–263.

FARBMAN, Z., HOFFER, G., LIPMAN, Y., COHEN-OR, D., AND LISCHINSKI, D. 2009. Coordinates for instant image cloning. In *SIGGRAPH '09: Proceedings of the 36th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA.

FATTAL, R., LISCHINSKI, D., AND WERMAN, M. 2002. Gradient domain high dynamic range compression. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 249–256.

FINLAYSON, G. D., HORDLEY, S. D., AND DREW, M. S. 2002. Removing shadows from images. In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part IV*, Springer-Verlag, London, UK, 823–836.

GIGAPAN, <http://www.gigapan.org/about.php>.

GORTLER, S., AND COHEN, M. 1995. Variational modeling with wavelets. In *Symposium on Interactive 3D graphics*, 35–42.

GRIEBEL, M., AND ZUMBUSCH, G. 1999. Parallel multigrid in an adaptive pde solver based on hashing and space-filling curves. *Parallel Comput.* 25, 7, 827–843.

HOCKNEY, R. W. 1965. A fast direct solution of Poisson's equation using Fourier analysis. *Journal of the ACM* 12, 1 (Jan.), 95–113.

HORN, B. K. P. 1974. Determining lightness from an image. *Comput. Graphics Image Processing* 3, 1 (Dec.), 277–299.

JIA, J., SUN, J., TANG, C.-K., AND SHUM, H.-Y. 2006. Drag-and-drop pasting. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, ACM, New York, NY, 631–637.

KAZHDAN, M., AND HOPPE, H. 2008. Streaming multigrid for gradient-domain operations on large images. *ACM Trans. Graph.* 27, 3, 1–10.

KAZHDAN, M., BOLITHO, M., AND HOPPE, H. 2006. Poisson surface reconstruction. In *Eurographics Symposium on Geometry Processing*, 61–70.

KAZHDAN, M. 2005. Reconstruction of solid models from oriented point sets. In *Eurographics Symposium on Geometry Processing*, 73–82.

KOPF, J., COHEN, M. F., LISCHINSKI, D., AND UYTENDAELE, M. 2007. Joint bilateral upsampling. *ACM Trans. Graph.* 26, 3, 96.

KOPF, J., UYTENDAELE, M., DEUSSEN, O., AND COHEN, M. F. 2007. Capturing and viewing gigapixel images. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, ACM, New York, NY, USA, 93.

LAWDER, J. K., AND KING, P. J. H. 2000. Using space-filling curves for multi-dimensional indexing. In *LNCS*, Springer Verlag, 20–35.

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

LISCHINSKI, D., FARBMAN, Z., UYTENDAELE, M., AND SZELISKI, R. 2006. Interactive local adjustment of tonal values. *ACM ToG* 25, 3, 646–653.

MCCANN, J., AND POLLARD, N. S. 2008. Real-time gradient-domain painting. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, ACM, New York, NY, USA, 1–7.

MCCANN, J. 2008. Recalling the single-FFT direct poisson solve. In *SIGGRAPH Posters*, ACM, 71.

MEGAPOV, <http://megapov.inetart.net>.

NASA, NASA Blue Marble <http://earthobservatory.nasa.gov/Features/BlueMarble/>.

NIEDERMEIER, R., REINHARDT, K., AND SANDERS, P. 1997. Towards optimal locality in meshindexings. In *Proc. Fundamentals of Computation Theory*, Springer, vol. 1279 of *LNCS*, 364–375.

PASCUCCI, V., AND FRANK, R. J. 2002. Hierarchical indexing for out-of-core access to multi-resolution data. In *Hierarchical and Geometrical Methods in Scientific Visualization*, Mathematics and Visualization. Springer, 225–241.

PÉREZ, P., GANGNET, M., AND BLAKE, A. 2003. Poisson image editing. *ACM Trans. Graph.* 22, 3, 313–318.

RICKER, P. M. 2008. A direct multigrid poisson solver for oct-tree adaptive meshes. *The Astrophysical Journal Supplement Series* 176, 293–300.

SAGAN, H. 1994. *Space-Filling Curves*. Springer-Verlag, New York, NY.

SIMCHONY, T., AND CHELLAPPA, R. 1990. Direct analytical methods for solving Poisson equations in computer vision problems. *IEEE Trans. Pattern Anal. Mach. Intell.* 12, 435–446.

SUN, J., JIA, J., TANG, C.-K., AND SHUM, H.-Y. 2004. Poisson matting. *ACM Trans. Graph.* 23, 3, 315–321.

SZELISKI, R. 2008. Locally adapted hierarchical basis preconditioning. *ACM Trans. Graph.* 27, 3, 1135–1143.

TOLEDO, S. 1999. A survey of out-of-core algorithms in numerical linear algebra. In *External memory algorithms*, Dimacs Series In Discrete Mathematics And Theoretical Computer Science. American Mathematical Society, Boston, MA, 161–179.

VITTER, J. S. 2001. External memory algorithms and data structures: dealing with massive data. *ACM Comput. Surv.* 33, 2, 209–271.

WEISS, Y. 2001. Deriving intrinsic images from image sequences. In *International Conference on Computer Vision*, 68–75.

A Hierarchical Z-Order

In this section we review the fundamentals of the *hierarchical Z-order (HZ-order)* for two-dimensional arrays, that is at the core of our data management scheme. We also provide a new, simple algorithm for accessing data organized in HZ-order, while avoiding the repeated index conversions used in [Pascucci and Frank 2002].

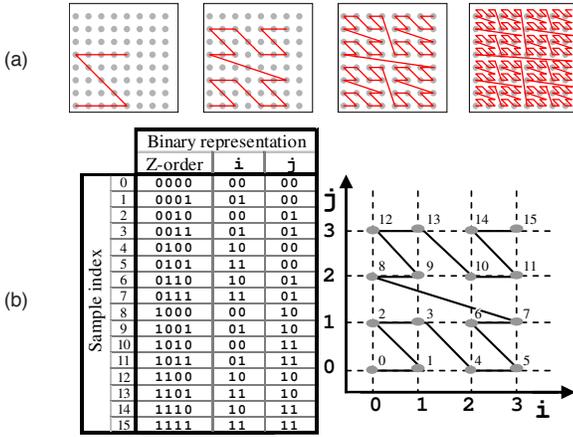


Figure 11: (a) The first four levels of the Z-order space filling curve; (b) 4x4 array indexed using standard Z-order

A.1 Previous Work

In the two-dimensional case the Z-order curve can be defined recursively by a Z shape whose vertices are replaced by Z shapes half its size (see Figure 6 (a)). Given the binary row-major index of a pixel $(i_n \dots i_1 i_0, j_n \dots j_1 j_0)$ the corresponding Z-order index I is computed by interleaving the indices $I = j_n i_n \dots j_1 i_1 j_0 i_0$ (see Figure ?? (a) step 1).

While Z-order exhibits good locality in all dimensions, it does so only at full resolution and does not support hierarchical access. Instead, our system uses the hierarchical variant, called HZ-order, proposed by Pascucci and Frank [2002]. This new index changes the standard Z-order of Figure 6 (b) to be organized by levels corresponding to a subsampling binary tree, in which each level doubles the number of points in one dimension (see Figure ?? (b)). This pixel order is computed by adding a second step to the index conversion. To compute an HZ-order index \hat{I} , the binary representation of a given Z-order index I is shifted to the right until the first 1-bit exits. During the first shift, a 1-bit is added to the left and 0-bits are added in all following shifts (see Figure ?? (a)).

We store the data in a way guaranteeing efficient access to any sub-region without internal caching and without opening a data block more than once. Furthermore, we allow for storage of incomplete arrays. In our storage format, we first sort the data in HZ-order and group consecutive samples in blocks of constant size. A sequence of consecutive blocks is grouped into a record and records are clustered in groups, which are organized hierarchically. Each record has a header specifying which of its blocks are actually present and if the data is stored raw or compressed. Groups can miss entire records or sub groups, implying that all their respective blocks and records are missing.

The file format is implemented via a header file describing the various parameters (dimension, block size, record size, etc.) and one file per record. The hierarchy of groups is implemented as a hierarchy of directories each containing a predetermined maximum number of subdirectories. The leaves of each directory contain only records. To open a file, one needs only to reconstruct the path of a record and defer its search to the file system. In particular, the path of a record is constructed as follows: we take the HZ-address of the first sample in the record, represent it as a string, and partition it into chunks of characters naming directories, subdirectories, and the record file. Note that, since blocks, records and groups can be missing, one is not restricted to arrays of data that cover the entire index space. In fact, we can easily store even images with different resolutions sampled at different resolutions.

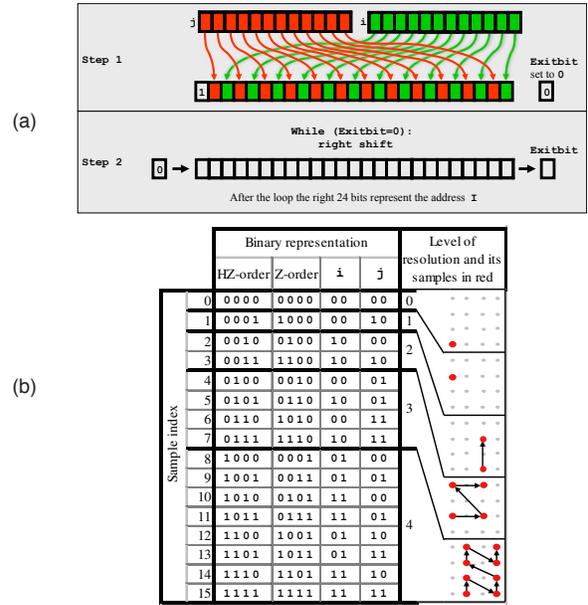


Figure 12: (a) Address transformation from row-major index (i, j) to Z-order index I (Step 1) and then to hierarchical Z-order index (Step 2); (b) Levels of the hierarchical Z-order for a 4x4 array. The samples on each level remain ordered by the standard Z-order.

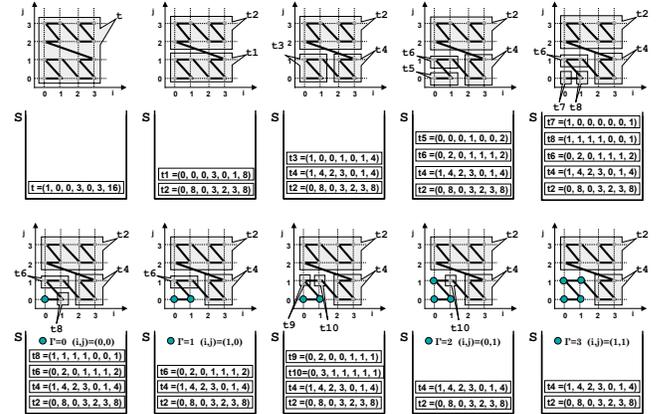


Figure 13: Our fast-stack Z-order traversal of a 4x4 array with concurrent index computation

A.2 Efficient Multi-Resolution Range Queries

One of the key components of our framework is the ability to quickly extract rectangular subsets of the input image in a progressive manner. Computing the row-major indices of all samples residing within a given query box is straightforward. However, efficiently calculating their corresponding HZ-indices is not. Transforming each address individually results in a large number of redundant computations by repeatedly converting similar indices. To avoid this overhead, we introduce a recursive access scheme that traverses an image in HZ-order, while concurrently computing the corresponding row-major indices. This traversal implicitly follows a kd-tree style subdivision, allowing us to quickly skip large portions of the image.

To better illustrate the algorithm we first describe how to recursively traverse an array in plain Z-order using the 4x4 array of Figure 6 (b) as example. Subsequently, we discuss how to restrict the traversal to a given query rectangle and finally how the scheme is adapted to HZ-order.

We use a stack containing tuples of type $(split_dimension, L_start, min_i, max_i, min_j, max_j, num_elements)$. To start the process we push the tuple $t_0 = (1, 0, 0, 3, 0, 3, 16)$ onto the stack. At each iteration we pop the top-most element t from the stack. If t contains only a single element we output the current L_start as HZ-index and fetch the corresponding sample. Otherwise, we split the region represented by t into two pieces along the axis given by $split_dimension$ and create the corresponding tuples $t_1 = (0, 0, 0, 3, 0, 1, 8)$ and $t_2 = (0, 8, 0, 3, 2, 3, 8)$. Note that all elements of t_1 and t_2 can be computed from t by simple bit manipulation. In case of a square array, we simply flip the split dimension each time a tuple is split. However, one can also store a specific split order to accommodate rectangular arrays. Figure ?? shows the first eight iteration of the algorithm outputting the first four elements in the array of Figure 6 (b).

To use this algorithm for fast range queries, each tuple is tested against the query box as it comes off the stack and discarded if no overlap exists. Since the row-major indices describing the bounding box of each tuple are computed concurrently, the intersection test is straightforward. Furthermore, the scheme applies, virtually unchanged, to traverse samples in Z-order that sub-sample an array uniformly along each axis, where the sub-sampling rate along each axis could be different.

Finally, to adapt the algorithm to HZ-order (see Figure ?? (b)), one exploits the following two important facts:

- One can directly compute the starting HZ-index for each level. For example, in a squared array level 0 contains one sample and all other levels h contain 2^{h-1} samples. Therefore the starting HZ-index of level h , I_{start}^h , is 2^{m-h} where m is the number of bits of the largest HZ-index.
- Within each level, samples are ordered according to plain Z-order and can be traversed with the stack algorithm described above, using the appropriate sub-sampling rate.

Using these two facts one can iterate through an array in HZ-order by processing one level at a time, adding I_{start}^h to the L_start index of each tuple.

In practice, we avoid subdividing the stack tuples to the level of a single sample. Instead, depending on the platform, we choose a parameter n and build a table, with the sequence of Z-order indices for an array with 2^n elements. When running the stack algorithm, each time a tuple t with 2^n elements appears, we loop through the table instead of splitting t . By accessing only the necessary samples in strict HZ-order, the stack-based algorithm guarantees that only the minimal number of disk blocks are touched and each block is loaded exactly once.

For progressively refined zooms in a given area, we can apply this algorithm with a minor variation. In particular, one would need to reduce the size of the bounding box represented in a tuple each time it is pushed back into the stack. In this way, even for a progressively refined zoom, one would access only the needed data blocks, each being accessed only once.