Compressive Rendering of Multidimensional Scenes

Pradeep Sen, Soheil Darabi, and Lei Xiao

Advanced Graphics Lab, University of New Mexico, Albuquerque, NM 87113

Abstract. Recently, we proposed the idea of using compressed sensing to reconstruct the 2D images produced by a rendering system, a process we called compressive rendering. In this work, we present the natural extension of this idea to multidimensional scene signals as evaluated by a Monte Carlo rendering system. Basically, we think of a distributed ray tracing system as taking point samples of a multidimensional scene function that is sparse in a transform domain. We measure a relatively small set of point samples and then use compressed sensing algorithms to reconstruct the original multidimensional signal by looking for sparsity in a transform domain. Once we reconstruct an approximation to the original scene signal, we can integrate it down to a final 2D image which is output by the rendering system. This general form of compressive rendering allows us to produce effects such as depth-of-field, motion blur, and area light sources, and also renders animated sequences efficiently.

1 Introduction

The process of rendering an image as computed by Monte Carlo (MC) rendering systems involves the estimation of a set of integrals of a multidimensional function that describes the scene. For example, for a scene with depth-of-field and motion blur, we can think of the distributed ray tracing system as taking point samples of a 5D continuous "scene signal" $f(x, y, u, v, t)$, where $f()$ is the scene-dependent function, (x, y) represents the position of the sample on the image, (u, v) is its position on the aperture for the depth-of-field, and t which describes the time at which the sample is calculated. The ray tracing system can compute point samples of this function by fixing the parameters (x, y, u, v, t) and evaluating the radiance of a ray with those parameters. The basic idea of Monte Carlo rendering is that by taking a large set of random point samples of this function, we can approximate the definite integral:

$$I(i,j) = \int_{j-\frac{1}{2}}^{j+\frac{1}{2}} \int_{i-\frac{1}{2}}^{i+\frac{1}{2}} \int_{t_0}^{t_1} \int_{-1}^{1} \int_{-1}^{1} f(x,y,u,v,t)\, du\, dv\, dt\, dx\, dy, \tag{1}$$

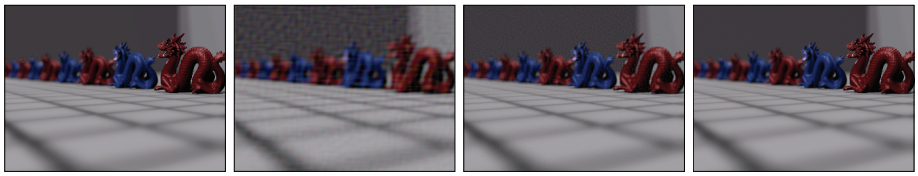
which gives us the value of the final image I at pixel (i, j) by integrating over the camera aperture from $[-1, 1]$, over the time that the shutter is open $[t_0, t_1]$, and over the pixel for antialiasing. In rendering, we use Monte Carlo integration

to estimate integrals like these because finding an analytical solution to these integrals is nearly impossible for real scene functions $f()$. Unfortunately, Monte Carlo rendering systems require a large number of multidimensional samples in order to converge to the actual value of the integral, because the variance of the estimate of the integral decreases as $O(1/k)$ with the number of samples k. If a small number of samples is used, the resulting image is very noisy and cannot be used for high-end rendering applications. The noise in the Monte Carlo result is caused by variance in the estimate, and there have been many approaches proposed in the past for reducing the variance in MC rendering.

One common method for variance reduction is *stratified sampling*, wherein the integration domain is broken up into a set of equally-sized non-overlapping regions (or strata) and a single sample is placed randomly in each, which reduces the variance of the overall estimate [1]. Other techniques for variance reduction exist, but they typically require more information about $f()$. For example, *importance sampling* positions samples with a distribution $p(x)$ that mimics $f()$ as closely as possible. It can be shown that if $p(x)$ is set to a normalized version of $f()$, then the variance of our estimator will be exactly zero [2]. However, this normalization involves knowing the integral of $f()$, which is obviously unknown in our case. Nevertheless, importance sampling can be useful when some information about the shape of $f()$ is known, such as the position of light sources in a scene. In this work, however, we assume that we do not know anything about the shape of $f()$ that we can use to position samples, which makes our approach a kind of technique often known as *blind* Monte Carlo. The only assumption we will make is that $f()$ is a real-world signal that is sparse or compressible in a transform domain.

Other kinds of variance reduction techniques have been proposed that introduce biased estimators, meaning that the expected value of the estimator is not equal to the exact value of the integral. Although methods such as stratified or importance sampling are both unbiased, biased Monte Carlo algorithms are also common in computer graphics (e.g., photon mapping [3]) because they sometimes converge much faster while yielding plausible results. The proposed approach in this paper also converges much more quickly than the traditional unbiased approaches, but it results in a slightly biased result. As we shall see, this occurs because of the discretization of the function when we pose it within the framework of compressed sensing (CS). However, this bias is small while the improvement in the convergence rate is considerable.

This chapter is based on ideas presented by the first two authors in work published in the IEEE Transactions of Visualization and Computer Graphics [4] and the Eurographics Rendering Symposium (EGSR) [5]. In the first work, we introduced the idea of using compressed sensing as a way of filling missing pixel information in order to accelerate rendering. In that approach, we first render only a fraction of the pixels in the image (which provides the speedup) and then we estimate the values of the missing pixels using compressed sensing by assuming that the final image is compressible in a transform domain. In the second work, we began to expand this idea to the concept of estimating an



(a) Original image **(b)** 2D sparsification **(c)** 3D sparsification **(d)** 4D sparsification

Fig. 1. Showing the effect on dimensionality on the compressibility of the signal in the Fourier domain. As the dimension of our scene function $f()$ increases, the compressibility of the data increases as well. Here we show a 4D scene with pixel antialiasing (2D) and depth-of-field (2D), which we have sparsified to 98% sparsity in the Fourier domain by zeroing out 98% of the Fourier coefficients. **(a)** Reference image. **(b)** Image generated by integrating down the function to 2D and then sparsifying it to 98% in the Fourier domain. We can see a significant amount of ringing and artifacts, which indicates that the 2D signal is not very compressible in the Fourier domain. This is the reason that we use wavelets for compression when handling 2D signals (see Secs. 5 and 6). **(c)** Image generated by integrating down the function to 3D (by integrating out the u parameter) and then sparsifying it to 98% in the Fourier domain. There are less artifacts than before, although they are still visible. **(d)** Image generated by sparsifying the original scene function $f()$ to 98% in the Fourier domain. The artifacts are greatly reduced here, indicating that as the dimensionality of the signal goes up the transform-domain sparsity also increases.

underlying multidimensional signal which we then integrate down to produce our final image. At the Dagstuhl workshop on Computational Video [6], we presented initial results on applying these ideas to animated video sequences (see Sec. 8).

This work presents a more general framework for compressive rendering that ties all of these ideas together into an algorithm that can handle a general set of Monte Carlo effects by estimating a multidimensional scene function from a small set of samples. By moving to higher-dimensional data sets, we improve the quality of the reconstructions because compressed sensing algorithms improve as the signal becomes more sparse, and as the dimension of the problem increases the sparsity (or technically, compressibility) of the signal also increases. The reason for this is that the amount of data in the signal goes up exponentially with the dimension, but the amount of actual information does not increase at this rate. As shown in Fig. 1, a 4D signal sparsified to 98% produces a much better quality image than a 2D signal sparsified the same amount.

To present this work, we first begin by describing previous work in rendering as it relates to Monte Carlo rendering and transform-domain accelerations proposed in the past. Next, we present a brief introduction to the theory of compressed sensing, since it is a field still relatively new to computer graphics. In Sec. 4, we present an overview of our general approach as well as a simple 1D example to compare the reconstruction of a signal from CS with those of traditional

techniques such as parametric fitting. Secs. 5 – 10 then show applications of this framework starting with 2D signals and building up to more complex 4D scenes. Finally, we end the chapter with some discussion and conclusions. We note that since this paper is in fact a generalization of two previous papers [4, 5], we have taken the liberty to heavily draw from our own text from these papers and the associated technical report [7], often verbatim, to maintain consistency across all the publications. We also duplicate results as necessary for completeness of this text.

2 Previous Work

Our framework allows to produce noise-free Monte Carlo rendered images with a small set of samples by filling in the missing samples of the multidimensional function using compressed sensing. Similar topics have been the subject of research in the graphics community for many years. We break up the previous work into algorithms that exploit transform-domain sparsity for rendering, algorithms that accelerate the rendering process outright, algorithms that are used to fill in missing sample information, and finally applications of compressed sensing in computer graphics.

2.1 Transform Compression in Rendering

There is a long history of research into transform-based compression to accelerate or improve rendering algorithms. We briefly survey some of the relevant work here and refer readers to more in-depth surveys (e.g., [8, 9]) for more detail. For background on wavelets, the texts by Stollnitz et al. [10] and Mallat [11] offer good starting points.

In the area of image rendering, transform compression techniques have been used primarily for accelerating the computation of illumination. For example, the seminal work of Hanrahan et al. [12] uses an elegant hierarchical approach to create a multiresolution model of the radiosity in a scene. While it does not explicitly use wavelets, their approach is equivalent to using a Haar basis. This work has been extended to use different kinds of wavelets or to subdivide along shadow boundaries to further increase the efficiency of radiosity algorithms, e.g., [13–15].

Recently, interest in transform-domain techniques for illumination has been renewed through research into efficient pre-computed radiance transfer methods using bases such as spherical harmonics [16, 17] or Haar wavelets [18, 19]. Again, these approaches focus on using the sparsity of the illumination or the BRDF reflectance functions in a transform domain, not on exploiting the sparsity of the final image.

In terms of using transform-domain approaches to synthesize the final image, the most successful work has been in the field of volume rendering. In this area, both the Fourier [20, 21] and wavelet domains [22] have been leveraged to reduce rendering times. However, the problem they are solving is significantly different than that of image rendering, so their approaches do not map well to

the problem addressed in this work. Finally, perhaps the most similar rendering approach is the frequency-based ray tracing work of Bolin and Meyer [23]. Like our own approach, they take a set of samples and then try to solve for the transform coefficients (in their case the Discrete Cosine Transform) that would result in those measurements. However, the key difference is that they solve for these coefficients using least-squares, which means that they can only reconstruct the frequencies of the signal that have sufficient measurements as given by the Nyquist-Shannon sampling theorem. Our approach, on the other hand, is based on the more recent work on compressed sensing, which specifies that the sampling rate is dependent on the sparsity of the signal rather than on its band-limit. This allows us to reconstruct frequencies higher than that specified by the Nyquist rate. We show an example of this in Sec. 4.1 that highlights this difference. By posing the problem of determining the value of missing samples within the framework of compressed sensing, we leverage the diverse set of tools that have been recently developed for these kinds of problems.

2.2 Accelerating ray tracing and rendering

Most of the work in accelerating ray tracing has focused on novel data structures for accelerating the scene traversal [24]. These methods are orthogonal to ours since we do not try to accelerate the ray tracing process (which involves point-sampling the multidimensional function) but rather focus on generating a better image with less samples. However, there are algorithms to accelerate rendering that take advantage of the spatial correlation of the final image, which in the end is related to the sparsity in the wavelet domain. Most common is the process of adaptive sampling [25, 26], in which a fraction of the samples are computed and new samples are computed only where the difference between measured samples is large enough, e.g., by a measure of contrast. Unlike our approach, however, adaptive sampling still computes the image in the spatial domain which makes it impossible to apply arbitrary wavelet transforms. For example, in the 2D missing pixel case of Sec. 5 we use the CDF 9/7 wavelet transform because it has been shown to be very good at compressing imagery. In other sections, we use sparsity in the Fourier domain. It is unclear how existing adaptive methods could be modified to use bases like this.

There is also a significant body of work which attempts to reconstruct images from sparse samples by using specialized data structures. First, there are systems which try to improve the interactivity of rendering by interpolating a set of sparse, rendered samples, such as the Render Cache [27] and the Shading Cache [28]. There are also approaches that perform interpolation while explicitly observing boundary edges to prevent blurring across them. Examples include Pighin et al.'s image-plane discontinuity mesh [29], the directional coherence map proposed by Guo [30], the edge-and-point image data structure of Bala et al. [31], and the real-time silhouette map by Sen [32, 33]. Our work is fundamentally different than these approaches because we never explicitly encode edges or use a data structure to improve the interpolation between samples. Rather, we take advantage of the compressibility of the final multidimensional signal in

a transform domain in order to reconstruct it and produce the final image. This allows us to faithfully reconstruct edges in the image, as can be seen by our results.

2.3 Reconstruction of missing data

Our approach only computes a fraction of the samples and uses compressed sensing to "guess" the values of the missing samples of the multidimensional scene function. In computer graphics and vision, many techniques have been proposed to fill in missing sample data. In the case of 2D signals such as images, techniques such as inpainting [34] and hole-filling [35] have been explored. Typically, these approaches work by taking a band of measured pixels around the unknown region and minimizing an energy functional that propagates this information smoothly into the unknown regions while at the same time preserving important features such as edges. Although we could use these algorithms to fill in the missing pixels in our 2D rendering application, the random nature of the rendered pixels makes our application fundamentally different from that of typical hole-filling, where the missing pixels have localized structure due to specific properties of the scene (such as visibility) which in our case are not available until render time. Furthermore, these methods become a lot less effective and more complex when trying to fill the missing data for higher dimensional cases, specially as we get to 4D scene functions or larger. Nevertheless, we compare our algorithm to inpainting in Sec. 5 to help validate our approach.

Perhaps the most successful approaches for reconstructing images from non-uniform samples for the 2D case come from the non-uniform sampling community, where this is known as the "missing data" problem since one is trying to reconstruct the missing samples of a discrete signal. Readers are referred to Ch. 6 of the principal text on the subject by Marvasti [36] for a complete explanation. One successful algorithm is known as ACT [37] which tries to fit trigonometric polynomials (related to the Fourier series) to the point-sampled measurements in a least-squares sense by solving the system using Toeplitz matrix inversion. This is related to the frequency-based ray tracing by Bolin and Meyer [23] described earlier. Another approach, known as the Marvasti method [38], solves the missing data problem by iteratively building up the inverse of the system formed by the non-uniform sampling pattern combined with a low pass filter. However, both the ACT and Marvasti approaches fundamentally assume that the image is bandlimited in order to do the reconstruction, something that is not true in our rendering application. As we show later in this paper, our algorithm relaxes the bandlimited assumption and is able to recover some of the high-frequency components of the image signal. Nevertheless, since ACT and Marvasti represent state-of-the-art approaches in the non-uniform sampling community for the reconstruction of missing pixels in a non-uniformly sampled image, we will compare our approach against these algorithms in Sec. 5. Unfortunately, neither of these algorithms is suitable for higher dimensional signals.

2.4 Compressed sensing and computer graphics

In this paper, we use tools developed for compressed sensing to solve the problem of reconstructing rendered images with missing pixel samples. Although compressed sensing has been applied to a wide range of problems in many fields, in computer graphics there are only a few published works that have used CS. Other than the work on compressive rendering on which this paper is based [4,5], most of the other applications of CS in graphics are in the area of light-transport acquisition [39–41]. The important difference between this application and our own is that it is not easy to measure arbitrary linear projections of the desired signal in rendering, while it is very simple to do so in light transport acquisition through structured illumination. In other words, since computing the weighted sum of a set samples is linearly harder than calculating the value of a single sample, our approach for a rendering framework based on compressive sensing had to be built around random point sampling. This will become more clear as we give a brief introduction to the theory of compressed sensing in the next section.

3 Compressed Sensing Theory

In this section, we summarize some of the key theoretical results of compressed sensing in order to explain our compressive rendering framework. A summary of the notation we shall use in this paper is shown in Table 1. Readers are referred to the key papers of Candès et al. [42] and Donoho [43] as well as the extensive CS literature available through the Rice University online repository [44] for a more comprehensive review of the subject.

3.1 Theoretical background

The theory of compressed sensing allows us to reconstruct a signal from only a few samples if it is sparse in a transform domain. To see how, suppose that we have an n-dimensional signal $\mathbf{f} \in \mathbb{R}^n$ we are trying to estimate with k random point samples, where $k \ll n$. We can write this sampling process with the linear sampling equation $\mathbf{y} = \mathbf{S}\mathbf{f}$, where $\mathbf{S}$ is an $k \times n$ sampling matrix that contains a single "1" in each row and no more than a single "1" in each column to represent each point-sampled value, and with zeros for all the remaining elements. This maps well to our rendering application, where the n-pixel image we want to render ($\mathbf{f}$) is going to be estimated from only k pixel samples ($\mathbf{y}$).

Initially, it seems that perfect estimation of $\mathbf{f}$ from $\mathbf{y}$ is impossible, given that there are $(n - k)$ pixels which we did not observe and could possibly have any value (ill-posedness). This is where we use the key assumption of compressed sensing: we assume that the image $\mathbf{f}$ is sparse in some transform domain $\hat{\mathbf{f}} = \mathbf{\Psi}^{-1}\mathbf{f}$. Mathematically, the signal $\hat{\mathbf{f}}$ is m-sparse if it has at most m non-zero coefficients (where $m \ll n$), which can be written in terms of the ℓ_0 norm (which effectively "counts" the number of non-zero elements): $\|\hat{\mathbf{f}}\|_0 \leq m$. This

n	size of final signal
k	number of evaluated samples
m	number of non-zero coefficients in transform domain
$\mathbf{f}$	high-resolution final signal, represented by an $n \times 1$ vector
$\hat{\mathbf{f}}$	transform of the signal, represented by a $n \times 1$, m-sparse vector
$\mathbf{y}$	$k \times 1$ vector of samples of $\mathbf{f}$ computed by the ray tracer
$\mathbf{S}$	$k \times n$ sampling matrix of the ray tracer, s.t. $\mathbf{y} = \mathbf{Sf}$
$\mathbf{\Psi}$	$n \times n$ "synthesis" matrix, s.t. $\mathbf{f} = \mathbf{\Psi}\hat{\mathbf{f}}$, and its associated inverse $\mathbf{\Psi}^{-1}$
$\mathbf{A}$	$k \times n$ "measurement" matrix, $\mathbf{A} = \mathbf{S\Psi}$

Table 1. Notation used in this paper.

is not an unreasonable assumption for real-world signals such as images, since this fact is exploited in transform-coding compression systems such as JPEG and MPEG. The basic idea of compressed sensing is that through this assumption, we are able to eliminate many of the images in the $(n-k)$-dimensional subspace which do not have sparse properties. To formulate the problem within the compressed sensing framework, we substitute our transform-domain signal $\hat{\mathbf{f}}$ into our sampling equation:

$$\mathbf{y} = \mathbf{Sf} = \mathbf{S\Psi}\hat{\mathbf{f}} = \mathbf{A}\hat{\mathbf{f}}, \tag{2}$$

where $\mathbf{A} = \mathbf{S\Psi}$ is a $k \times n$ *measurement* matrix that includes both the sampling and compression bases. If we could solve this linear system correctly for $\hat{\mathbf{f}}$ given $\mathbf{y}$, we could then recover the desired $\mathbf{f}$ by taking the inverse transform. Unfortunately, solving for $\hat{\mathbf{f}}$ is difficult to do with traditional techniques such as least squares because the system is severely undetermined because $k \ll n$. However, one of the key results in compressed sensing demonstrates that if $k \geq 2m$ and the Restricted Isometry Condition (RIC) condition is met (Sec. 3.4), then we can solve for $\hat{\mathbf{f}}$ uniquely by searching for the sparsest $\hat{\mathbf{f}}$ that solves the equation. A proof of this remarkable conclusion can be found in the paper by Candès et al. [42]. Therefore, we can pose the problem of computing the transform of the final rendered image from a small set of samples as the solution of the ℓ_0-optimization problem:

$$\min \|\hat{\mathbf{f}}\|_0 \text{ s.t. } \mathbf{y} = \mathbf{A}\hat{\mathbf{f}}. \tag{3}$$

Unfortunately, algorithms to solve Eq. 3 are NP-hard [45] because they involve a combinatorial search of all m-sparse vectors $\hat{\mathbf{f}}$ to find the sparsest one that meets the constraint. Fortunately, the CS research community has developed fast algorithms that find approximate solutions to this problem. In this paper we use solvers such as ROMP and SpaRSA to compute the coefficients of signal $\hat{\mathbf{f}}$ within the context of compressed sensing. We give an overview of these algorithms in the following two sections.

3.2 Overview of ROMP algorithm

Since the solution of the ℓ_0 problem in Eq. 3 requires a brute-force combinatorial search of all the the $\hat{\mathbf{f}}$ vectors with sparsity less than m, the CS research community has been developing fast, greedy algorithms that find approximate solutions

Algorithm 1 ROMP algorithm

Input: measured vector $\mathbf{y}$, matrices $\mathbf{A}$ and $\mathbf{A}^\dagger$, target sparsity m
Output: the vector $\hat{\mathbf{f}}$, which is an m-sparse solution of $\mathbf{y} = \mathbf{A}\hat{\mathbf{f}}$
Initialize: $I = \emptyset$ and $\mathbf{r} = \mathbf{y}$

while $\mathbf{r} \neq \mathbf{0}$ and sparsity not met **do**
  $\mathbf{u} \Leftarrow \mathbf{A}^\dagger \mathbf{r}$ /* *multiply residual by* $\mathbf{A}^\dagger$ *to approx. larger coeffs of* $\hat{\mathbf{f}}$ */
  $J \Leftarrow$ sort coefficients of $\mathbf{u}$ in non-increasing order
  $J_0 \Leftarrow$ contiguous set of coefficients in J with maximal energy
  $I \Leftarrow I \cup J_0$ /* *add new indices to overall set* */
  /* *find vector of* I *coeffs that best matches measurement* */
  $\hat{\mathbf{f}}_{new} \Leftarrow \underset{\mathbf{z}\,:\,\mathrm{supp}\,(\mathbf{z})=I}{\mathrm{argmin}} \|\mathbf{y} - \mathbf{A}\mathbf{z}\|_2$
  $\mathbf{r} \Leftarrow \mathbf{y} - \mathbf{A}\hat{\mathbf{f}}_{new}$ /* *recompute residual* */
end while
return $\hat{\mathbf{f}}_{new}$

to the ℓ_0 problem. One example is Orthogonal Matching Pursuit (OMP) [46], which iteratively attempts to find the non-zero elements of $\hat{\mathbf{f}}$ one at a time. To do this, OMP is given the measured vector $\mathbf{y}$ and measurement matrix $\mathbf{A}$ as input and it finds the coefficient of $\hat{\mathbf{f}}$ with the largest magnitude by projecting $\mathbf{y}$ onto each column of $\mathbf{A}$ through the inner product $|\langle \mathbf{y}, \mathbf{a}_j \rangle|$ (where $\mathbf{a}_j$ is the j^{th} column of $\mathbf{A}$) and selecting the largest. After the largest coefficient has been identified, we assume that this is the only non-zero coefficient in $\hat{\mathbf{f}}$ and approximate its value by solving the $\mathbf{y} = \mathbf{A}\hat{\mathbf{f}}$ using least-squares. The new estimate for $\hat{\mathbf{f}}$ with a single non-zero coefficient is then used to compute the estimated signal $\mathbf{f}$, which is subtracted from the original measurements to get a residual. The algorithm then iterates again, using the residual to solve for the next largest coefficient of $\hat{\mathbf{f}}$, and so on. It continues to do this until an m-sparse approximation of the transform domain vector is found.

Despite its simplicity, OMP has a weaker guarantee of exact recovery than the ℓ_1 methods [47]. For this reason, Needell and Vershynin proposed a modification to OMP called Regularized Orthogonal Matching Pursuit (ROMP) which recovers multiple coefficients in each iteration, thereby accelerating the algorithm and making it more robust to meeting the RIC. Essentially, ROMP approximates the largest-magnitude coefficients of $\hat{\mathbf{f}}$ in a similar way to OMP, by projecting $\mathbf{y}$ onto each column of $\mathbf{A}$ and sorts them in decreasing order. It then finds all of the continuous sets of coefficients in this list whose largest coefficient is at most twice as big as the smallest member, and selects the set with the maximal energy. These indices are added to a list that is maintained by the algorithm which keeps track of the non-zero coefficients of $\hat{\mathbf{f}}$, and the values of those coefficients are computed by solving $\mathbf{y} = \mathbf{A}\hat{\mathbf{f}}$ through least-squares assuming that these are the only non-zero coefficients. As in OMP, the new estimate for $\hat{\mathbf{f}}$ is then used to compute the estimated signal $\mathbf{f}$ which is subtracted from $\mathbf{y}$ to get a residual. The algorithm continues to iterate using the residual as the input and solving for the next largest set of coefficients of $\hat{\mathbf{f}}$ until an m-sparse approximation of

the transform domain vector is found, an error criteria is met, or the number of iterations exceeds a certain limit without convergence. Although the ℓ_1 problem requires $N = O(m \log k)$ samples to be solved uniquely, in practice we find that the ROMP algorithm requires around $N = 5m$ samples to start locking in to the correct solution and $N = 10m$ to work extremely robustly. Since we use the ROMP algorithm in both 2D signal reconstruction applications (see Secs. 5 and 6), we provide a pseudocode description for reference in Alg. 1.

3.3 Overview of SpaRSA algorithm

Another algorithm we use in this paper is known as SpaRSA. One of the key results of recent compressed sensing theory is that the problem of Eq. 3 can be framed as an ℓ_1 problem instead, where the ℓ_1 norm is defined as the sum of the absolute values of the elements of the vector ($\|\mathbf{v}\|_1 = \sum_{i=1}^{k} |v_i|$):

$$\min \|\hat{\mathbf{f}}\|_1 \quad \text{s.t. } \mathbf{y} = \mathbf{A}\hat{\mathbf{f}}. \tag{4}$$

Candès et al. [42] showed that this equation has the same solution as Eq. 3 if $\mathbf{A}$ satisfies the RIC and the number of samples $N = O(m \log k)$, where m is the sparsity of the signal. This fundamental result spurred the flurry of research in compressed sensing, because it demonstrated that these problems could be solved by tractable algorithms such as linear programming. Unfortunately, it is still difficult to solve the ℓ_1 problem, so researchers in applied mathematics have been working on novel algorithms to provide a fast solution. One successful avenue of research is to reformulate Eq. 4 into what is known as the $\ell_2 - \ell_1$ problem:

$$\min_{\hat{\mathbf{f}}} \frac{1}{2}\|\mathbf{y} - \mathbf{A}\hat{\mathbf{f}}\|_2^2 + \tau\|\hat{\mathbf{f}}\|_1. \tag{5}$$

In this formulation, the first term enforces the fit of the solution to the measured values $\mathbf{y}$ while the second term looks for the smallest ℓ_1 solution (and hence the sparsest solution). The parameter τ balances the optimization towards one constraint or the other. Recently, Wright et al. proposed a novel solution to Eq. 5 by solving a simple iterative subproblem with an algorithm they call Sparse Reconstruction by Separable Approximation (SpaRSA) [48]. In this work, we use SpaRSA to reconstruct scene signals $\mathbf{f}$ that are 3D and larger. Unfortunately, even a simple explanation of the SpaRSA is beyond the scope of this paper. Interested readers are referred to the technical report associated with our EGSR paper [7] for more information.

3.4 Restricted Isometry Condition (RIC)

It is impossible to solve $\mathbf{y} = \mathbf{A}\hat{\mathbf{f}}$ for *any* arbitrary $\mathbf{A}$ if $k \ll n$ because the system is severely underdetermined. However, compressed sensing can be used to solve uniquely for $\hat{\mathbf{f}}$ if matrix $\mathbf{A}$ meets the Restricted Isometry Condition (RIC):

$$(1 - \epsilon)||\mathbf{v}||_2 \leq ||\mathbf{A}\mathbf{v}||_2 \leq (1 + \epsilon)||\mathbf{v}||_2, \tag{6}$$

with parameters (z, ϵ), where $\epsilon \in (0, 1)$ for all z-sparse vectors $\mathbf{v}$ [47]. Effectively, the RIC states that in a valid measurement matrix $\mathbf{A}$, every possible set

of z columns of $\mathbf{A}$ will form an approximate orthogonal set. Another way to say this is that the sampling and compression bases $\mathbf{S}$ and $\mathbf{\Psi}$ that make up $\mathbf{A}$ must be incoherent. Examples of matrices that have been proven to meet RIC include Gaussian matrices (with entries drawn from a normal distribution), Bernoulli matrices (binary matrices drawn from a Bernoulli distribution), and partial Fourier matrices (randomly selected Fourier basis functions) [49].

In this paper, we can use point-sampled Fourier (partial Fourier matrices) for all of the applications with a scene function 3D or higher, but for the 2D cases where we are reconstructing an image, the Fourier basis does not provide enough compression (as shown in Fig. 1) and cannot be used. In this case, we would like to use the wavelet transform which does provide enough compression, but unfortunately a point-sampled wavelet basis does not meet the RIC. We discuss our modifications to the wavelet basis to improve this and allow our framework to be used for the reconstruction of 2D scene functions in Sec. 5. With this theoretical background in place, we can now give an overview of our algorithm and a simple 1D example.

4 Algorithm overview

The basic idea of the proposed rendering framework is quite simple. We use a distributed ray tracing system that takes a small set of point samples of the multidimensional scene function $f()$. In traditional Monte Carlo, these samples would be added together to estimate the integral of $f()$, but in our case assume that the signal $f()$ is sparse in a transform domain and use the compressed sensing theory described in the previous section to estimate a discrete reconstruction of $f()$. This reconstructed version is then integrated down to form our final image.

Since the CS solvers operate on discrete vector and matrices, we first approximate the unknown function $f()$ with a discrete vector $\mathbf{f}$ of size n by taking uniform samples of $f()$. This approximation is reasonable as long as n is large enough, since it is equivalent to discretizing $f()$. For example, if we assume that the signal $f()$ is sparse in the Fourier basis composed of 2π-periodic basis functions, then the signal $f()$ must also be 2π periodic. Therefore, the samples that form $\mathbf{f}$ must cover the 2π interval of $f()$ to ensure periodicity and therefore maintain the sparsity in the Discrete Fourier Transform domain. In this case, for example, the i^{th} component of $\mathbf{f}$ is given by $f_i = f(\frac{2\pi}{k}(i-1))$. By sampling the function $f()$ in this manner when discretizing it, we guarantee that $\mathbf{f}$ will also be sparse in transform domain Ψ, where the columns of Ψ are discrete versions of the basis functions ψ_i from the equations above. Note that we do not explicitly sample $f()$ to create $\mathbf{f}$ (since we do not know $f()$ a priori), but rather we assume a n-length vector $\mathbf{f}$ exists which is the discrete version of the unknown $f()$ and which we will solve for through CS.

We can now take our k random measurements of $\mathbf{f}$, as given by $\mathbf{y} = \mathbf{Sf}$ where $\mathbf{S}$ is the $k \times n$ sampling matrix, by point-sampling the original function $f()$ at the appropriate discrete locations. Therefore, unlike traditional Monte Carlo

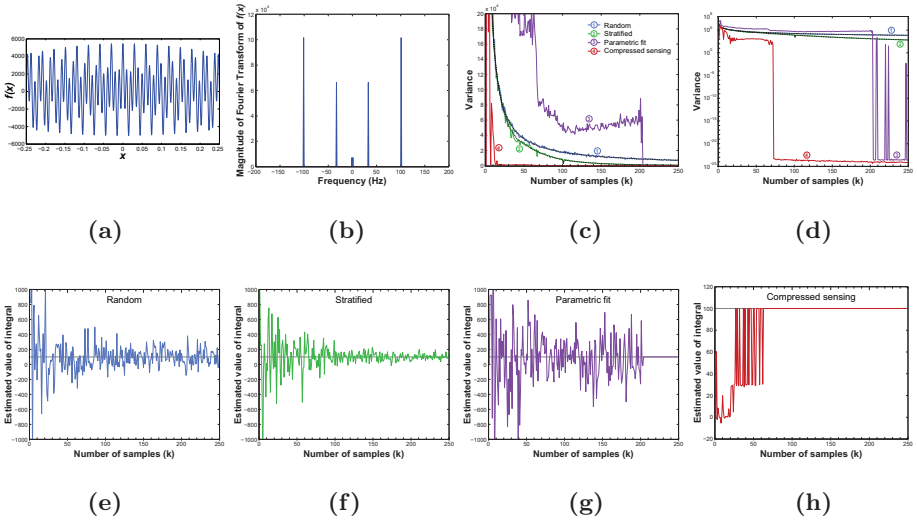


Fig. 2. Results of the simple example of Sec. 4.1. (a) Plot of the signal $f(x)$ which we want to integrate over the interval shown. **(b)** Magnitude of the Fourier Transform of $f(x)$. Since 3 cosines of different frequencies are added together to form $f(x)$, its Fourier Transform has six different spikes because of symmetry (so sparsity $m = 6$). **(c)** Linear and **(d)** log plots of variance as a function of the number of samples to show the convergence of the four different integration algorithms. The faint dashed lines in the "random" and "stratified" curves (visible in the pdf) show the theoretical variance, which matches the experimental results. The log plot clearly shows the "waterfall" curve characteristic of compressed sensing reconstruction, where once the adequate number of samples are taken the signal is reconstructed perfectly every time. In this case, we need around 70 samples, which is roughly $10\times$ the number of spikes in **(b)**. **(e − h)** Plots of the estimated value of the integral vs. the number of samples for one run with the different integration algorithms. The correct value of the integral (100) is shown as a gray line. We can see that the compressed sensing begins to approximate the correct solution around $k = 5m = 30$ samples and then "snaps" to the right answer at for $k > 10m = 60$, which is much more quickly than the other approaches.

approaches, our random samples do not occur arbitrarily along the continuous domain of $f()$, but rather at a set of discrete locations that represent the samples of $\mathbf{f}$. Once the set of samples that form measurement vector $\mathbf{y}$ have been taken, we use compressed sensing to solve for the coefficients that correspond to the non-zero basis functions $\hat{\mathbf{f}}$. We can then take the inverse transform $\mathbf{f} = \mathbf{\Psi}\hat{\mathbf{f}}$ and integrate it down to get our final image. To help explain how our algorithm works, we now look at a simple 1D example.

4.1 Example of 1D Signal Reconstruction

At first glance, it might seem that what we are proposing to do is merely just another kind of parametric fit, somehow fitting a function to our samples to

approximate $f()$. Although we are fitting a function to the measured data, compressed sensing offers us a fundamentally different way to do this than the traditional methods used for parametric fitting, such as least-squares. We can see the difference with a simple 1D example. Suppose we want to compute the definite integral from $-\frac{1}{4}$ to $\frac{1}{4}$ of the following function $f(x)$, which is unknown to us *a priori* but is shown in Fig. 2**(a)**:

$$f(x) = \alpha[a\pi\cos(2\pi ax)] + \beta[b\pi\cos(2\pi bx)] + \gamma[c\pi\cos(2\pi cx)].$$

The signal is made up of three pure frequencies, and in this experiment we set $a = 1$, $b = 33$, and $c = 101$ so that there is a reasonable range of frequencies represented in the signal. This particular function is constructed so that the analytic integral is easy to compute; the integral of each of the terms in square brackets over the specified interval is equal to 1. This means that in this case, the desired definite integral is

$$I = \int_{-\frac{1}{4}}^{\frac{1}{4}} f(x)\,dx = \alpha + \beta + \gamma.$$

In this experiment, we set $\alpha = 70$, $\beta = 20$, and $\gamma = 10$, so that the desired integral $I = 100$. Within the context of our rendering problem, we assume that do not know $f(x)$ in analytical form, so our goal is to compute the integral value of 100 simply from a set of random point samples of $f(x)$.

The most common way to do this in computer graphics is to use Monte Carlo integration, which takes k uniformly-distributed, random samples over the entire interval $[-\frac{1}{4}, \frac{1}{4}]$ and use these to estimate the integral. Although the answer fluctuates based on the position of our measurements, as we add more and more samples the estimator slowly converges to the correct answer as can be seen in the variance curves of Fig. 2**(c, d)** and the results of a single run while varying the number of samples in **(e)**. We can compute the theoretical variance of a random Monte Carlo approach analytically which gives us a theoretical variance shown in Fig. 2**(c, d)** as a thin, dashed line. We can see that the theoretical variance calculated matches well with the experimental results.

The slow $\frac{1}{k}$ decay of variance with random Monte Carlo is less than desirable for rendering applications, so a common variance-reduction technique is stratified sampling. Fig. 2**(f)** shows the result of one run with this method, and indeed we notice that the estimate of the integral gets closer to the correct solution (shown by a thin gray line) more quickly than with the random approach. The theoretical variance of the stratified approach can be computed in software by computing the variance of each of the strata for every size k. The resulting curve also matches the experimental results, even predicting a small dip in variance around $k = 101$. However, stratified sampling still takes considerable time to converge, so it is worthwhile to examine other techniques that might be better.

Another way we might consider computing the integral of this function from the random samples is to try to fit a parametric model to the measurements and then perform the integral on the model itself. Indeed, both our approach and the parametric fit approach require us to know something about the signal (e.g., that it can be compactly represented with sinusoids). However, closer observation reveals that our framework is based on fundamentally different theory than

typical methods for fitting parametric models and so it yields a considerably different result.

To see why, let us work through the process of actually fitting a parametric sinusoidal model to our measured samples. Typically, this involves solving a least-squares problem which in this case means solving $\mathbf{y} = \mathbf{A}\hat{\mathbf{f}}$ for the coefficients of $\hat{\mathbf{f}}$, where $\mathbf{A} = \mathbf{S}\boldsymbol{\Psi}$ is the sampling matrix multiplied by the Fourier basis. Because we are solving the problem with least squares, we need to have a "thin" matrix (or at least square) for $\mathbf{A}$, which means that the number of unknown coefficients in $\hat{\mathbf{f}}$ can be at most k, matching the number of observations at $\mathbf{y}$. Since the Fourier transform of $\mathbf{f}$ has two sets of complex conjugate elements in $\hat{\mathbf{f}}$, the highest frequency we can solve for uniquely in this manner is at most $k/2$. This traditional approach is closely related to the Nyquist-Shannon sampling theorem [50], well-known in computer graphics, which states that to correctly reconstruct a signal we must have sampled at more than twice its highest frequency.

Indeed, trying to fit a parametric model in this traditional manner means that we only get correct convergence of the integral we have more samples than twice the highest frequency $c = 101$, or around $k = 202$. As can be seen in Fig. 2**(g)**, the estimated value of the integral bounces around with fairly high variance until this point and then locks down to $I = 100$ when we start having enough samples to fit the sinusoids correctly. However, this process is not scalable, since it is dependent on the highest frequency of the signal. If we set $c = 1,000$, we would need ten times more samples to converge correctly. Our compressed sensing approach, on the other hand, has the useful property that its behavior is independent of the highest frequency.

In another approach, we might consider solving the parametric fit problem using a "fat" $\mathbf{A}$ matrix, so that the number of unknown elements in $\hat{\mathbf{f}}$, which is n, can be much bigger than k. Perhaps this will allow us to solve for higher frequency sinusoids even though we do not have enough samples. In this case the problem is under-determined and there could be many solutions with the same square error. Traditionally, these kinds of problems are solved with a least-norm algorithm, which finds the least-squares solution that also has the least norm in the ℓ_2 sense (where ℓ_2 is the square-root of the sum of the squares of the components). In our case the least-norm solution is given by $\hat{\mathbf{f}}_{ln} = \mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{y}$. Unfortunately, this works even worse than the least-squares fit for our example. It turns out that there are many "junk" signals that have a lower ℓ_2 norm than the true answer (because they contain lots of small values in their frequency coefficients instead of a few large ones), yet they still match the measured values in the least-squares sense.

Although these traditional methods for fitting parametric models are commonly used in computer graphics, they do not work for this simple example because they are dependent on the frequency content of our signal. The theory of compressed sensing, on the other hand, offers us new possibilities since it states that the number of samples is dependent on the *sparsity* of the signal, not the particular frequency content it may have. In this example, we can apply CS

to solve for $\mathbf{y} = \mathbf{A}\hat{\mathbf{f}}$ and then integrate the signal as discussed in the previous section. The results shown in Fig. 2**(h)** show that we "snap" to the answer after about 70 samples, which makes sense since we have observed empirically that ROMP typically requires 10 times more samples than the sparsity m, which is 6 in this case. At this point, the estimated value of the integral is 100.0030, where the 0.003 error is caused by the discretization of $\mathbf{f}$ as compared to $f(x)$. This is the bias of our estimator, but it is considerably small especially considering the value of the integral we are computing. Finally, we note that CS is able to reconstruct the discrete signal $\mathbf{f}$ perfectly and consistently with $k = 75$, even though we have *less than one sample per period* of the highest frequency ($c = 101$) in the signal, well below the Nyquist limit.

Before we finish this discussion we should mention some of the implementation details used to acquire the results of Fig. 2. While the random Monte Carlo and stratified experiments sample the analytic versions of $f(x)$ directly to compute the integral, the parametric and CS approaches need to solve linear systems of equations and therefore use the discrete version $\mathbf{f}$. For the CS experiments we set the size of $\mathbf{f}$ and $\hat{\mathbf{f}}$ to be $n = 2^{12} + 1$, which means that the highest frequency that we can solve for is 2048Hz. This is not a limiting factor, however, since we can increase this value (e.g., our later experiments for antialiasing in Sec. 6 use $n = 2^{20}$). When measuring the variance curves for random Monte Carlo, stratified sampling, and parametric fit, we ran 250 trials for each k to reduce their noisiness. The compressed sensing reconstruction was more consistent and we only used 75 trials for each k to compute its variance. Note that the variance of the CS approach is down to the 10^{-24} range when it has more than 70 samples, which after 75 independent trials means that the value of the integral is rock solid and stable at this point.

This simple example might help motivate our approach in theory, but we need to validate our framework by using it to solve a real-world problem in computer graphics. We spend the rest of the chapter discussing the application of this framework to a set of different problems in rendering.

5 Application to 2D signals – Image reconstruction

In our first example, we begin by looking at the problem of image reconstruction from a subset of pixels. The basic idea is to accelerate the rendering process by simply computing a subset of pixels and then reconstructing the missing pixels using the ones we measured. We plan to use compressed sensing to do this by looking for the pixel values that would create the sparsest signal possible in a transform domain.

In this case, since we are working with 2D images, the suitable compression basis would be wavelets. Unfortunately, although wavelets are very good at compressing image data, they are incompatible with the point-sampling basis of our rendering system because they are not incoherent with point samples as required by Restricted Isometry Condition (RIC) of compressed sensing. To see why, we note that the coherence between a general sampling basis $\mathbf{\Omega}$ and compression

basis $\mathbf{\Psi}$ can be found by taking the maximum inner product between any two basis elements of the two:

$$\mu(\mathbf{\Omega}, \mathbf{\Psi}) = \sqrt{n} \cdot \max_{1 \leq j,k \leq n} |\langle \omega_j, \psi_k \rangle|. \tag{7}$$

Because the matrices are orthonormal, the resulting coherence lies in the range $\mu(\mathbf{\Omega}, \mathbf{\Psi}) \in [1, \sqrt{n}]$ [51], with a fully incoherent pair having a coherence of 1. This is the case for the point-sampled Fourier transform, which is ideal for compressed sensing but unfortunately is not suitable for our application because of its lack of compressibility for 2D images as shown in Fig. 1. If we use a wavelet as the compression basis (e.g., a $64^2 \times 64^2$ Daubechies-8 wavelet (DB-8) matrix for $n = 64^2$), the coherence with a point-sampled basis is 32, which is only half the maximum coherence possible ($\sqrt{n} = 64$). This large coherence makes wavelets unsuitable to be used as-is in the compressed sensing framework.

In order to reduce coherency yet still exploit the wavelet transform, we propose a modification to Eq. 2. Specifically, we assume that there exists a blurred image $\mathbf{f}_b$ which can be sharpened to form the original image: $\mathbf{f} = \mathbf{\Phi}^{-1}\mathbf{f}_b$, where $\mathbf{\Phi}^{-1}$ is a sharpening filter. We can now write the sampling process as $\mathbf{y} = \mathbf{Sf} = \mathbf{S\Phi}^{-1}\mathbf{f}_b$. Since the blurred image $\mathbf{f}_b$ is also sparse in the wavelet domain, we can incorporate the wavelet compression basis in the same way as before and get $\mathbf{y} = \mathbf{S\Phi}^{-1}\mathbf{\Psi}\hat{\mathbf{f}}_b$. We can now solve for the sparsest $\hat{\mathbf{f}}_b$:

$$\min \|\hat{\mathbf{f}}_\mathbf{b}\|_0 \text{ s.t. } \mathbf{y} = \mathbf{A}\hat{\mathbf{f}}_\mathbf{b}, \tag{8}$$

where $\mathbf{A} = \mathbf{S\Phi}^{-1}\mathbf{\Psi}$, using the greedy algorithms such as OMP or ROMP. Once $\hat{\mathbf{f}}_b$ has been found, we can compute our final image by taking the inverse wavelet transform and sharpening the result: $\mathbf{f} = \mathbf{\Phi}^{-1}\mathbf{\Psi}\hat{\mathbf{f}}_b$.

In this work, our filter $\mathbf{\Phi}$ is a Gaussian filter, and since we can represent the filtering process as multiplication in the frequency domain, we write $\mathbf{\Phi} = \mathcal{F}^H\mathbf{G}\mathcal{F}$, where $\mathcal{F}$ is the Fourier transform matrix and $\mathbf{G}$ is a diagonal matrix with values of a Gaussian function along its diagonal. Substituting this in to Eq. 8, we get:

$$\min \|\hat{\mathbf{f}}_\mathbf{b}\|_0 \text{ s.t. } \mathbf{y} = \mathbf{S}\mathcal{F}^H\mathbf{G}^{-1}\mathcal{F}\mathbf{\Psi}\hat{\mathbf{f}}_\mathbf{b}. \tag{9}$$

We observe that $\mathbf{G}^{-1}$ is also a diagonal matrix and should have the values $\mathbf{G}_{i,i}^{-1}$ along its diagonal. However, we must be careful when inverting the Gaussian function because it is prone to noise amplification. To avoid this problem, we use a linear Wiener filter to invert the Gaussian [52], which means that the diagonal elements of our inverse matrix $\mathbf{G}^{-1}$ have the form $\mathbf{G}_{i,i}^{-1} = \mathbf{G}_{i,i}/(\mathbf{G}_{i,i}^2 + \lambda)$. Since the greedy algorithms (such as ROMP) we use to solve Eq. 3 require a "backward" matrix $\mathbf{A}^\dagger$ that "undoes" the effect of $\mathbf{A}$ (i.e., $\|\mathbf{A}^\dagger\mathbf{Av}\| \approx \|\mathbf{v}\|$), where $\mathbf{A} = \mathbf{S\Phi}^{-1}\mathbf{\Psi}$, we use a backwards matrix of the form $\mathbf{A}^\dagger = \mathbf{\Psi}^{-1}\mathbf{\Phi}\mathbf{S}^T = \mathbf{\Psi}^{-1}\mathcal{F}^H\mathbf{G}\mathcal{F}\mathbf{S}^T$.

Note that for real image sizes, the matrix $\mathbf{A}$ will be too large to store in memory. For example, to render a 1024×1024 image with a 50% sampling rate, our measurement matrix $\mathbf{A}$ will have $k \times n = 5.5 \times 10^{11}$ elements. Therefore, our

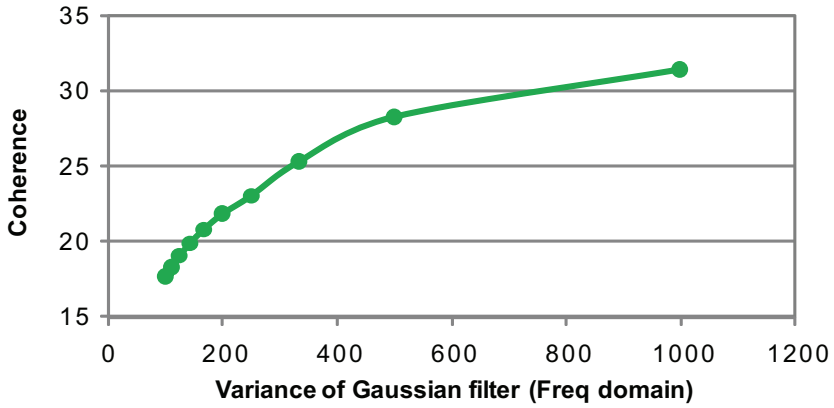


Fig. 3. Coherence vs. variance of Gaussian matrix $\mathbf{G}$. Since $\mathbf{G}$ is in the frequency domain, larger variance means a smaller spatial filter. As the variance grows, the coherence converges to 32, the coherence of the point-sampled, $64^2 \times 64^2$ DB-8 matrix. The coherence should be as small as possible, which suggests a smaller variance for our Gaussian filter in the frequency domain. However, this results in a blurrier image $\mathbf{f}_b$ which is harder to reconstruct accurately. The optimal values for the variance were determined empirically and are shown in Table 2 for different sampling rates.

implementation must use a functional representation for $\mathbf{A}$ that can compute the required multiplications such as $\mathbf{A}\hat{\mathbf{f}}_\mathbf{b}$ on the fly as needed.

The addition of the sharpening filter means that our measurement matrix is composed of two parts: the point-samples $\mathbf{S}$ and a "blurred wavelet" matrix $\mathbf{\Phi}^{-1}\mathbf{\Psi}$ which acts as the compression basis. This new compression basis can be thought of as either blurring the image and then taking the wavelet transform, or applying a "filtered wavelet" transform to the original image. To see how this filter reduces coherence, we plot the result of Eq. 7 as a function of the variance σ^2 of Gaussian function of $\mathbf{G}$ in Fig. 3 for our $64^2 \times 64^2$ example. Note that the Gaussian $\mathbf{G}$ is in the frequency domain, so as the variance gets larger the filter turns into a delta function in the spatial domain and the coherence approaches 32, the value of the unfiltered coherence. As we reduce the variance of $\mathbf{G}$, the filter gets wider in the spatial domain and coherence is reduced by almost a factor of 2.

Although it would seem that the variance of $\mathbf{G}$ should be as small as possible (lowering coherence), this increases the amount of blur of $\mathbf{f}_b$ and hence the noise in our final result due to inversion of the filter. We determined the optimal variances empirically on a single test scene and used the same values for all our experiments (see Table 2). In the end, the reduction of coherence by a factor of 2 through the application of the blur filter was enough to yield good results with compressed sensing.

To test our algorithm, we integrated our compressive sensing framework into both an academic ray tracing system (PBRT [24]) and a high-end, open source ray tracer (LuxRender [53]). The integration of both was straightforward since

%	$1/\sigma^2$	λ	%	$1/\sigma^2$	λ
6%	0.000130	0.089	53%	0.000015	0.259
13%	0.000065	0.109	60%	0.000014	0.289
25%	0.000043	0.209	72%	0.000013	0.289
33%	0.000030	0.209	81%	0.000011	0.299
43%	0.0000225	0.234	91%	0.000008	0.399

Table 2. Parameters for the Gaussian ($1/\sigma^2$) and Wiener filters (λ). We iterated over the parameters of the Gaussian filter to find the ones that yielded the best reconstruction for the ROBOTS scene at the given sampling rates (%) for 1024×1024 reconstruction.

we only had to control the pixels being rendered (to compute only a fraction of the pixels), and then add on a reconstruction module that performed the ROMP algorithm. In order to select the random pixels to measure, we used the Boundary sampling method of the Poisson-disk implementation from Dunbar and Humphreys [54] to space out the samples in image-space. For LuxRender, for example, these positions were provided to the ray tracing system through the PixelSampler class. The "low discrepancy" sampler was used to ensure that samples were only made in the pixels selected. After the ray tracer evaluated the samples, the measurement was recorded into a data structure that was fed into the ROMP solver. The rest of the ray tracing code was left untouched.

The ROMP solver was based on the code by Needell et al. [47] available on their website but re-written in C++ for higher performance. We leverage the Intel Math Kernel Library 10.1 (MKL) libraries [55] to accelerate linear algebra computation and to perform the Fast-Fourier Transform for our Gaussian filter. In addition, we use the Stanford LSQR solver [56] to solve the least-squares step at the end of ROMP. The advantage of LSQR is that it is functional-based so we do not need to represent the entire $\mathbf{A}$ matrix in memory since it can get quite large as mentioned earlier.

To describe the implementation of the functional version of the measurement matrix $\mathbf{A}$, we first recall that $\mathbf{A} = \mathbf{S}\mathcal{F}^H\mathbf{G}^{-1}\mathcal{F}\mathbf{\Psi}$ from Eq. 9. The inverse wavelet transform $\mathbf{\Psi}$ was computed using the lifting algorithm [11], and the MKL library was used to compute the Fourier and inverse-Fourier transforms of the signal. To apply the filter, we simply weighted the coefficients by the Gaussian function described in the algorithm. After applying the inverse Fourier transform to the filtered signal, we then simply take the samples from the desired positions. This gives us a way to simulate the effect of matrix $\mathbf{A}$ in our ROMP algorithm without explicitly specifying the entire matrix. In addition, we found empirically that ROMP behaved better when the maximum number of coefficients added in each iteration was bounded. For the experiments in this chapter, we used a bound of $2k/i$, where k is the number of pixels measured and i is the maximum number of ROMP iterations which we set to 30.

After the renderer finishes computing the samples, ROMP operates on the input vector $\mathbf{y}$. We set the target sparsity to one fifth of the number of samples

Scene	Pre-process	Full Render	CS Recon	%
Robots	0.25	611	11.9	1.9%
Watch	0.28	903	13.5	1.4%
Sponza	47	634	12.0	1.9%

Table 3. Timing results in minutes of our algorithm. **Pre-process** includes loading the models, creating the acceleration data structures. **Full render** is the time to sufficiently sample every pixel to generate the ground-truth image. **CS Recon** is the time it took our reconstruction algorithm to solve for a 1024×1024 image with 75% of samples. The last column shows the percentage of rays that could be traced instead of using our approach. Because our CS reconstruction is fast, this number is fairly small. We ignore post-processing effects because these are in the order of seconds and are negligible. We also do not include the cost of the interpolation algorithm since it took around 10 seconds to triangulate and interpolate the samples.

k, which has been observed to work well in the CS literature [57]. ROMP uses the Gaussian filter of Eq. 9 as part of the reconstruction process. The parameters of the Gaussian filter were set through iterative experiments on a single scene, but once set they were used for all the scenes in this chapter. Table 2 shows the actual values used in our experiments. If a sampling rate is used that is not in the table, the nearest entries are interpolated. The compression basis used in this work is the Cohen-Daubechies-Feauveau (CDF) 9-7 biorthogonal wavelet, which is particularly well-suited for image compression and is the wavelet used in JPEG2000 [58]. Since we are dealing with color images, the image signal must be reconstructed in all three channels: R, G, and B. To accelerate reconstruction, we transform the color to YUV space and use the compressive rendering framework for only the Y channel and use the Delaunay-interpolation (described below) for the other two. The error introduced by doing this is not noticeable as we are much more sensitive to the Y channel in an image than the other the two.

To compare our results, we test our approach against a variety of other algorithms that might be used to fill in the missing pixel data in the renderer. For example, we compare against the popular inpainting method of Bertalmio et al. [34], using the implementation by Alper and Mavinkurve [59]. To compare against approaches from the non-uniform sampling community, we implemented the Marvasti algorithm [38] and used the MATLAB version for the ACT algorithm provided in the *Nonuniform Sampling* textbook [36]. Finally, rendering systems in practice typically use interpolation methods to estimate values between computed samples. Unfortunately, many of the convolution-based methods that work so well for uniform sampling simply do not work when dealing with non-uniform sample reconstruction (see, e.g., the discussion by Mitchell [26]). For this work we implemented the piece-wise cubic multi-stage filter described by Mitchell [26], with the modification that we put back the original samples at every stage to improve performance. Finally, we also implemented the most common interpolation algorithm used in practice, which uses Delaunay triangulation to mesh the samples and then evaluates the color of the missing pixels in

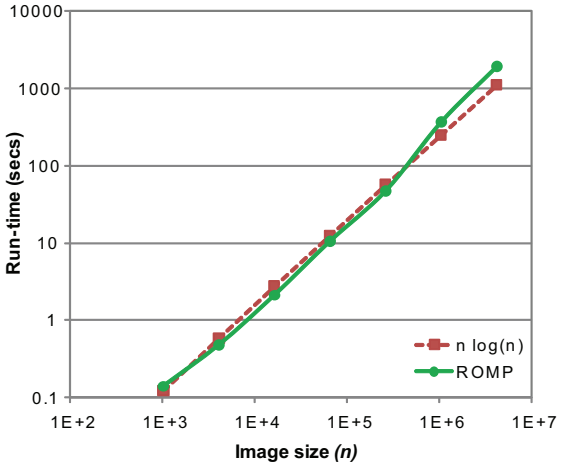


Fig. 4. Run-time complexity of our ROMP reconstruction algorithm, where n is the total size of our image (width $\times$ height). The curve of $n \log n$ is shown for comparison. We tested our algorithm on images of size 32×32 all the way through 2048×2048. Even at the larger sizes, the performance remained true to the expected behavior. Note that the complexity of the reconstruction algorithm is independent of scene complexity.

between by interpolating each triangle of the mesh, e.g., as described by Painter and Sloan [60]. This simple algorithm provides a piece-wise linear reconstruction of the image, which turned out to be one of the better reconstruction techniques.

The different algorithms were all tested on a Dell Precision T3400 with a quad-core, 3.0 Ghz Intel Core2 Extreme CPU QX6850 CPU with 4GB RAM capable of running 4 threads. The multi-threading is used by LuxRender during pixel sampling and by the Intel MKL library when solving the ROMP algorithm during reconstruction. Since most of the reconstruction algorithms have border artifacts, we render a larger frame and crop out a margin around the edges. For example, the 1000×1000 images were rendered at 1024×1024 with a 12-pixel border.

5.1 Timing performance

In order for the proposed framework to be useful, the CS reconstruction of step 2 has to be fast and take less time than the alternative of simply brute-force rendering more pixels. Table 3 shows the timing parameters of various scenes rendered with LuxRender. We see that the CS step takes approximately 10 minutes to run for a 1024×1024 image with 75% samples with our unoptimized C implementation. Since the full-frame rendering times are on the order of 6 to 15 hours, the CS reconstruction constitutes less than 2% of the total rendering time, which means that in the time to run our reconstruction algorithm only 2% of extra pixels could be computed. On the other hand, the inpainting implemen-

Scene	Interp	Cubic	ACT	Marvasti	Inpaint	CS
Robots	2.00	4.94	1.98	2.11	4.24	1.72
Watch	6.42	11.00	6.15	6.68	15.00	5.34
Sponza	2.38	6.77	2.38	2.65	4.70	2.11

Table 4. MSE performance of the various algorithms ($\times 10^{-4}$) for different scenes. All scenes were sampled with 60% of pixel samples.

tation we tested took approximately one hour to compute the missing pixels, making our ROMP reconstruction reasonably efficient by comparison.

We also examine the run-time complexity of the ROMP reconstruction as a function of image size to see how the processing times would scale with image size for images from 32×32 to 2048×2048 (see Fig. 4). We can see that it behaves as $O(n \log n)$ as predicted by the model. For the image sizes that we are dealing with ($\leq 10^7$ pixels) this is certainly acceptable, given the improvement in image quality we get with our technique. Finally, we note that our algorithm runs in image-space so it is completely independent of scene complexity. On the other hand, rendering algorithms scale as $O(n)$, but the constants involved depend on scene complexity and have a significant impact on the rendering time. Over the past few decades, feature film rendering times have remained fairly constant as advances in hardware and algorithms are offset by increased scene complexity. Since our algorithm is independent of the scene complexity, it will continue to be useful in the foreseeable future.

5.2 Image quality

Standard measures for image quality are typically ℓ_2 distance measures. In this work, we use the mean squared error (MSE) assuming that the pixels in the image have a range of 0 to 1, and compare the reconstructed images from all the approaches to the ground-truth original. Table 4 shows the MSE for the various algorithms we tested: the first two are interpolation algorithms, followed by the two algorithms from the non-uniform sampling community, then the result of inpainting and finally the result of the CS-based reconstruction proposed in this work. Our algorithm has the lowest MSE, something that we observed in all our experiments.

A few additional points are worth mentioning. First of all, we noticed that the inpainting algorithm performed fairly poorly in our experiments. The reason for this is that in our application the holes are randomly positioned, while these techniques require bands of known pixels around the hole (i.e., spatial locality). Unfortunately, this is not easy to do in a rendering system since we cannot cluster the samples a priori without knowledge of the resulting image. Also disappointing was Mitchell's multistage cubic filter, which tended to overblur the image when we set the kernel large enough to bridge the larger holes in the image. Although the algorithms from the non-uniform sampling community (ACT and Marvasti) perform better, they are on par with the Delaunay-interpolation used in rendering which works remarkably well.

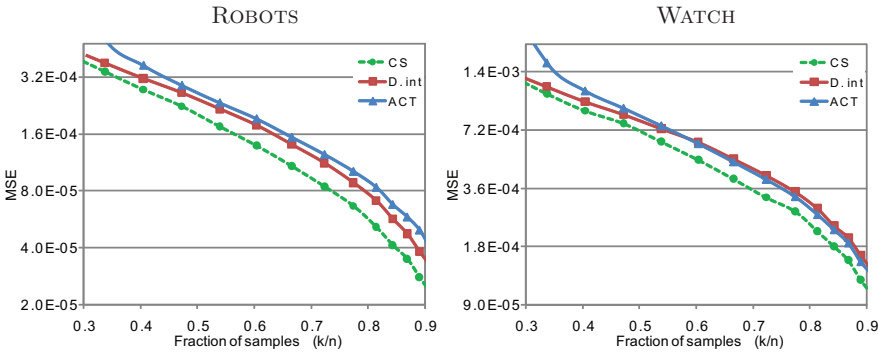


Fig. 5. Log error curves as a function of the number of samples for four test scenes using our technique and the best two other competing reconstruction algorithms. Our CS reconstruction beats both Delaunay-interpolation (D. int) as well as ACT, requiring 5% to 10% less samples to achieve a given level of quality.

To see how our algorithm would work at different sampling rates, we compare it against the two best competing methods (Delaunay-Interpolation and ACT) for two of our scenes in Fig. 5. We observe that to achieve a given image quality, Delaunay interpolation and ACT require about 5% to 10% more samples than our approach. When the rendering time is 10 hours, this adds up to an hour of savings to achieve comparable quality. Furthermore, since our algorithm is completely independent of scene complexity, the benefit of our approach over interpolation becomes more significant as the rendering time increases.

However, MSE is not the best indicator for visual quality, which is after all the most important criterion in high-end rendering. To compare the visual quality of our results, we refer the readers to Fig.12 at the end of the chapter. We observe that compressive rendering performs much better than interpolation in regions with sharp edges or those that are slightly blurred, a good property for a rendering system. To see this, we direct readers to the second inset of the ROBOTS scene in Fig. 12. Although the fine grooves in the robot's arm cannot be reconstructed faithfully by any of the other algorithms, compressed sensing is able to do this by selecting the values for the missing pixel locations that yield a sparse wavelet representation.

Another good example can be found on the third row of the WATCH scene. Here there is a pixel missing between two parts of the letter "E," which our algorithm is the only one to be able to correctly reconstruct. All other techniques simply interpolate between the samples on either side of the missing pixel and fill in this sample incorrectly. However, since clean, straight lines are more sparse than the jumbled noise estimated by the other approaches, they are selected by our technique. Although the ACT algorithm performs reasonably well overall, it suffers from ringing when the number of missing samples is high and there is a sharp edge (see e.g., the last inset of WATCH) because of the fitting of trigonometric polynomials to the point samples.

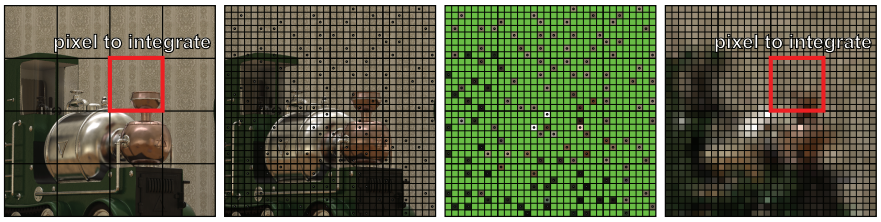


Fig. 6. Illustration of our antialiasing algorithm. (a) Original continuous signal $f(x)$ to be antialiased over the 4×4 pixel grid shown. **(b)** In our approach, we first take k samples of the signal aligned on an underlying grid of fixed resolution n. This is equivalent of taking k random samples of discrete signal $\mathbf{f}$. **(c)** The measured samples form our vector of measurements $\mathbf{y}$, with the unknown parts of $\mathbf{f}$ shown in green. Using ROMP, we solve $\mathbf{y} = \mathbf{A}\hat{\mathbf{f}}$ for $\hat{\mathbf{f}}$, where $\mathbf{A} = \mathbf{S}\boldsymbol{\Psi}$. $\mathbf{S}$ is the sampling matrix corresponding to the samples taken, $\boldsymbol{\Psi}$ is the blurred-wavelet basis described in Sec. 5. **(d)** Our approximation to $\mathbf{f}$, computed by applying the synthesis basis to $\hat{\mathbf{f}}$ (i.e., $\mathbf{f} = \boldsymbol{\Psi}\hat{\mathbf{f}}$). We integrate this approximation over each pixel to get our antialiased result.

6 Application to 2D signals – Antialiasing

In this section, we present another example of 2D scene reconstruction by applying our framework to the problem of box-filtered antialiasing. The basic idea is simple (see overview in Fig. 6). We first take a few random point samples of the scene function $f()$ per pixel. Unlike the previous section, we are no longer dealing with pixels of the image but rather samples on an underlying grid of higher resolution than the image that matches the size of the unknown discrete function $\mathbf{f}$ and is aligned with its samples. We then use ROMP to approximate a solution to Eq. 3 which can then be used to calculate $\mathbf{f}$. Once we have $\mathbf{f}$, we can integrate it over the pixel to perform our antialiasing.

The observation is that if $\mathbf{f}$ is sparse in the transform domain, we will need only a small set of samples to evaluate this integral accurately. Fig. 7 shows a visual comparison of our approach against stratified and random supersampling which are also used for antialiasing images. This is very similar to our previous approach, except now we have introduced the notion of applying integration to the reconstructed function to produce the final image.

7 Application to 3D signals – Motion Blur

We now describe the application of our framework to the rendering of motion blur, which involves the reconstruction of a 3D scene. Motion blur occurs in dynamic scenes when the projected image changes as it is integrated over the time the camera aperture is open. Traditionally, Monte Carlo rendering systems emulate motion blur by randomly sampling rays over time and accumulating them together to estimate the integral [61]. Conceptually, our approach is very similar to that of our antialiasing algorithm. We first take a set of samples of

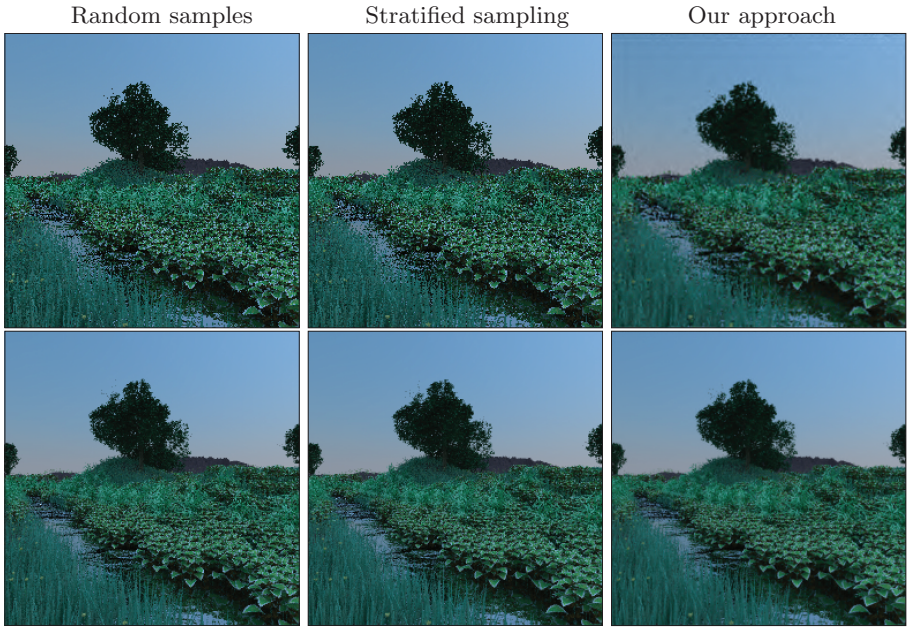


Fig. 7. Visual comparison for GARDEN **scene.** Each row has a different number of samples/pixel (from top to bottom: 1, 4).

the scene $\mathbf{y}$, except that now the measurements are also spaced out in time to sample the discrete spatio-temporal volume $\mathbf{f}$, which represents a set of video frames over the time the aperture was open. We then use compressed sensing to reconstruct $\hat{\mathbf{f}}$, the representation of the volume in the 3D Fourier transform domain $\mathbf{\Psi}$. After applying the inverse transform to recover an approximation to the original $\mathbf{f}$, we can then integrate it over time to achieve our desired result. An example of the computed motion blur is shown in Fig. 8.

8 Application to 3D signals – Video

An obvious extension to the motion blur application of the previous section is to view the individual frames of the reconstructed spatio-temporal volume directly, resulting in an algorithm to render animated sequences. This is a significant advantage over the more complex adaptive approaches that have been proposed for rendering (e.g., MDAS [62] or AWR [63]) which are difficult to extend to animated scenes because of their complexity. This is the reason that these previous approaches have dealt exclusively with the rendering of static imagery. To generate an animated sequence, these approaches render a set of static frames by evaluating each frame independently and do not take into account their temporal coherence. Our approach, on the other hand, uses compressed sensing to evaluate a sparse version of the signal in x, y, and t in the 3D Fourier domain, so it fully computes the entire spatio-temporal volume which we can view as frames in the video sequence.

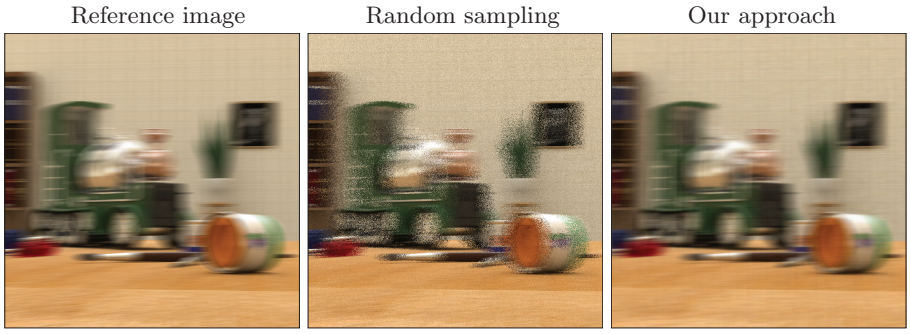


Fig. 8. Visual comparison of motion blur results for the TRAIN **scene.** The reference image was rendered with 70 temporal samples/pixel, while the other two where rendered with a single random sample per pixel in time. Our result was reconstructed assuming a spatio-temporal volume of 24 frames. Images were rendered at a resolution of 1000×1000.

To demonstrate this, we show individual frames in Fig. 9 from the dynamic TRAIN scene of Fig. 8. For comparison, we implemented an optimized linear interpolation using a 3-D Gaussian kernel, which is the best convolution methods can do when the sampling rate is so low. We also compare against our earlier compressive rendering work [5], which reconstructs the set of static images individually using sparsity in the wavelet domain. The second column of Fig. 9 shows the missing samples in green, which results in an image that is almost entirely green when we have a 1% sampling rate (only 1/100 pixels in the spatio-temporal volume are calculated). It is remarkable that our algorithm can reconstruct a reasonable image even at this extremely low sampling density, while the other two approaches fail completely.

This suggests that our technique could be useful for pre-visualization, since we get a reasonable image with $100\times$ less samples. The reconstruction time is less than a minute per frame using the unoptimized C++ SpaRSA implementation, while the rendering time is 8 minutes per frame on an Intel Xeon 2.93 GHz CPU-based computer with 16 GB RAM. This means that the ground-truth reference 128-frame video would take over 17 hours to compute. On the other hand, using our reconstruction with only 1% of samples we get a video with a frame shown in Fig. 9 (right image) in less than 2 hours.

9 Application to 4D signals – Depth of Field

We now show the application of our framework to a 4D scene function to demonstrate the rendering of depth-of-field. Monte Carlo rendering systems compute depth-of-field by estimating the integral of the radiance of incoming rays over the aperture of the lens through a set of random points on the virtual lens [61]. This means that we must choose two additional random parameters for each sample which tell us the position the ray passes through the virtual lens.

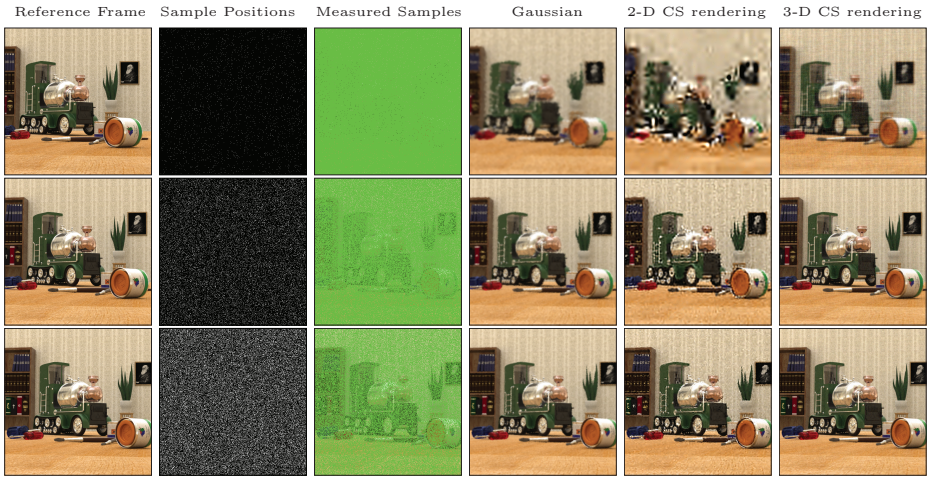


Fig. 9. Reconstructing frames in an animated sequence. We can use our approach to render individual frames in an animated sequence and leverage the coherence both spatially and temporally in the Fourier domain. The three rows represent different frames in the train sequence with the sampling rate varied for each (1% for the top row, 10% for the middle, and 25% for the bottom). The first column shows the reference frame of the fully-rendered sequence, the second column shows the positions of the samples (shown in white), the third column shows the samples available for this particular frame (unknown samples shown in green), the fourth column shows a reconstruction by convolving the samples with an optimized 3-D Gaussian kernel with variance adjusted for the sampling rate (best possible linear filter), the fifth column shows the results of reconstructing each frame separately using the 2D algorithm from Sec. 5, and the last column is reconstructs the entire 3-D volume using a 3D Fourier transform. These images are rendered at 512×512 with 128 frames in the video sequence.

Therefore, in this application we parameterize each sample by its image-space coordinate (x, y) and these two additional parameters (u, v). As mentioned in the previous sections, since our compressed sensing reconstruction works on discrete positions, we uniformly choose the positions on the virtual lens to lie on a grid. The proposed framework is general and therefore easy to map to this new problem. We take our sample measurements $\mathbf{y}$ by sampling this new 4D space, and then reconstruct the whole space $\mathbf{f}$ using compressed sensing by assuming the sparsity in the 4D Fourier domain. For this example, we use the SpaRSA solver to compute the sparse transform-domain signal $\hat{\mathbf{f}}$. The final image is calculated in the end by integrating the reconstructed 4D signal over all u and v for each pixel. Fig. 10 shows an example of the output of algorithm for the depth-of-field.

10 Application to 4D signals – Area Light Source

We can also apply the proposed framework to reconstruct a 4D scene with an area light source. This extension is fairly similar to that of the depth-of-field

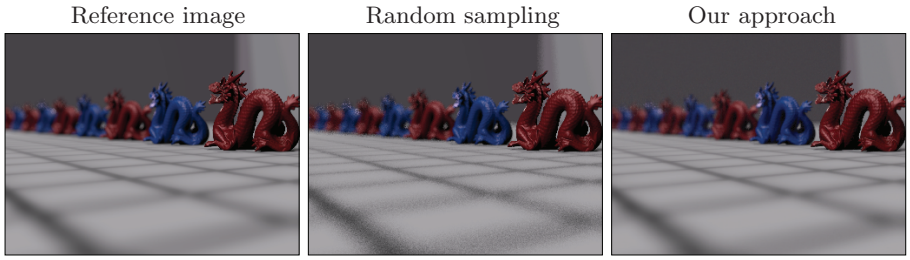


Fig. 10. Visual comparison of depth-of-field results for the DRAGON scene. The reference image was rendered with 256 samples per pixel. Our result was generated by reconstructing a signal of size $340 \times 256 \times 16 \times 16$ and integrating the 16×16 samples over (u, v) for each of the 340×256 pixels.

effect, with the only difference in that here the two random variables represent points on the area light source. In this way, we parametrize each sample by its image-space coordinate (x, y) along with the position on the area light source (p, q). Again, we sample this 4D space and reconstruct it with SpaRSA assuming the sparsity in the 4D Fourier domain. Finally, we integrate over the area light coordinates p and q for each pixel in the reconstructed space to determine the final pixel color. Fig. 11 shows an example of the result of the algorithm for an area light source.

11 Discussion

The framework proposed in this chapter is fairly general and as shown in these last few sections it can reconstruct a wide range of Monte Carlo effects. However, there are some issues that currently affect its practical use for production rendering. One of the current limiting factors for the performance of the system is the speed of the reconstruction by the solver. In this work, we used C++ implementations of ROMP [47] and SpaRSA [48], but these were still relatively slow (requiring tens of minutes for some of the reconstructions) which decreases the performance of the overall system. However, the applied mathematics community is constantly developing new CS solvers, and there are already solvers that appear to be much faster than ROMP or SpaRSA that we are just starting to experiment with. There is also a possibility to implement the CS solver on the GPU, which would give us a further speed up for our algorithm.

Another issue is the memory usage of the algorithm. Currently, the solvers need to store the entire signal $\mathbf{f}$ (or its transform $\hat{\mathbf{f}}$) in memory while the solver is calculating its components. As the dimension of our scene function grows, the size of $\mathbf{f}$ grows exponentially. For example, if we want to do compressive rendering for a 6D scene with depth-of-field and an area light source, say with an image resolution of 1024×1024 and sample grids of 16×16 for a lens and an area light source, we would have to store an $\mathbf{f}$ of size $n = 2^{36}$ entries, which would require 256 GB of memory. Therefore, our current approach suffers from the

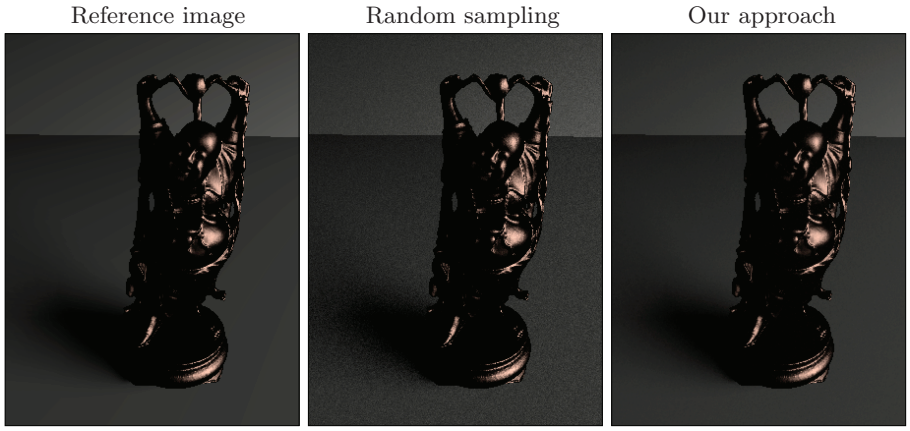


Fig. 11. Visual comparison of area light source results for the BUDDHA scene. The reference image was rendered with 256 samples per pixel. Our result was generated by reconstructing a signal of size $300 \times 400 \times 16 \times 16$ and integrating the 16×16 samples over (p, q) for each of the 300×400 pixels.

"curse of dimensionality" that can plague other approaches for multidimensional signal integration. We are currently working on modifying the solvers to ease the memory requirements of the implementation.

Nevertheless, this chapter presents a novel way to look at the Monte Carlo rendering by treating it as a multidimensional function that we can reconstruct fully by assuming that it is sparse in a transform domain using the tools from compressed sensing. This work might encourage other researchers to explore new ways to solve the rendering problem.

12 Conclusion

In this chapter, we have presented a general framework for compressive rendering that shows how we can use a distributed ray tracing system to take a small set of point samples of a multidimensional function $f()$, which we then can approximately reconstruct using compressed sensing algorithms such as ROMP and SpaRSA by assuming sparsity in a transform domain. After reconstruction, we can then integrate the signal down to produce the final rendered image. This algorithm works for a general set of Monte Carlo effects, and we demonstrate results with motion-blur, depth-of-field, and area light sources.

Acknowledgments

The authors would like to thank Dr. Yasamin Mostofi for fruitful discussions regarding this work. Maziar Yaesoubi, Nima Khademi Kalantari, and Vahid Noormofidi also helped to acquire some of the results presented. The DRAGON (Figs. 1

and 10), Buddha (Fig. 11), Sponza (Fig. 12) and Garden (Fig. 7) scenes are from the distribution of the PBRT raytracer by Pharr and Humphreys [24]. The Robot (Fig. 12) and Train (Fig. 6, 8 and 9) scenes are from J. Peter Lloyd, and the Watch (Fig. 12) scene is from Luca Cugia. This work was funded by the NSF CAREER Award #0845396 "A Framework for Sparse Signal Reconstruction for Computer Graphics."

References

1. Veach, E.: Robust monte carlo methods for light transport simulation. PhD thesis, Stanford University, Stanford, CA, USA (1998) Adviser-Leonidas J. Guibas.
2. Dutré, P., Bala, K., Bekaert, P., Shirley, P.: Advanced Global Illumination. AK Peters Ltd (2006)
3. Jensen, H.W.: Realistic image synthesis using photon mapping. A. K. Peters, Ltd., Natick, MA, USA (2001)
4. Sen, P., Darabi, S.: Compressive rendering: A rendering application of compressed sensing. IEEE Transactions on Visualization and Computer Graphics **17** (2011) 487–499
5. Sen, P., Darabi, S.: Compressive estimation for signal integration in rendering. Computer Graphics Forum (Proceedings of Eurographics Symposium on Rendering (EGSR) 2010) **29** (2010) 1355–1363
6. Sen, P., Darabi, S.: Exploiting the sparsity of video sequences to efficiently capture them. In Magnor, M., Cremers, D., Zelnik-Manor, L., eds.: Dagstuhl Seminar on Computational Video. (2010)
7. Sen, P., Darabi, S.: Details and implementation for compressive estimation for signal integration in rendering. Technical Report EECE-TR-10-0003, University of New Mexico (2010)
8. Schröder, P.: Wavelets in computer graphics. Proceedings of the IEEE **84** (1996) 615–625
9. Schröder, P., Sweldens, W.: Wavelets in computer graphics. SIGGRAPH 1996 Course Notes (1996)
10. Stollnitz, E.J., Derose, T.D., Salesin, D.H.: Wavelets for computer graphics: theory and applications. Morgan Kaufmann Publishers Inc. (1996)
11. Mallat, S.: A Wavelet Tour of Signal Processing. Second edn. Academic Press (1999)
12. Hanrahan, P., Salzman, D., Aupperle, L.: A rapid hierarchical radiosity algorithm. SIGGRAPH Comput. Graph. **25** (1991) 197–206
13. Gortler, S.J., Schröder, P., Cohen, M.F., Hanrahan, P.: Wavelet radiosity. In: SIGGRAPH '93. (1993) 221–230
14. Lischinski, D., Tampieri, F., Greenberg, D.P.: Combining hierarchical radiosity and discontinuity meshing. In: SIGGRAPH. (1993) 199–208
15. Schröder, P., Gortler, S.J., Cohen, M.F., Hanrahan, P.: Wavelet projections for radiosity. Computer Graphics Forum **13** (1994)
16. Ramamoorthi, R., Hanrahan, P.: An efficient representation for irradiance environment maps. In: SIGGRAPH. (2001)
17. Sloan, P.P., Kautz, J., Snyder, J.: Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In: SIGGRAPH. (2002) 527–536

18. Ng, R., Ramamoorthi, R., Hanrahan, P.: All-frequency shadows using non-linear wavelet lighting approximation. ACM Trans. Graph. **22** (2003) 376–381
19. Ng, R., Ramamoorthi, R., Hanrahan, P.: Triple product wavelet integrals for all-frequency relighting. In: SIGGRAPH. (2004) 477–487
20. Malzbender, T.: Fourier volume rendering. ACM Trans. Graph. **12** (1993) 233–250
21. Totsuka, T., Levoy, M.: Frequency domain volume rendering. In: SIGGRAPH. (1993) 271–278
22. Gross, M.H., Lippert, L., Dittrich, R., Hring, S.: Two methods for wavelet-based volume rendering. Computers and Graphics **21** (1997) 237 – 252
23. Bolin, M., Meyer, G.: A frequency based ray tracer. In: SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques. (1995) 409–418
24. Pharr, M., Humphreys, G.: Physically Based Rendering: From Theory to Implementation. Morgan Kaufmann Publishers Inc. (2004)
25. Whitted, T.: An improved illumination model for shaded display. Communications of the ACM **23** (1980) 343–349
26. Mitchell, D.P.: Generating antialiased images at low sampling densities. In: SIGGRAPH. (1987) 65–72
27. Walter, B., Drettakis, G., Parker, S.: Interactive rendering using the render cache. In Lischinski, D., Larson, G., eds.: Proceedings of the 10th Eurographics Workshop on Rendering. Volume 10., New York, NY, Springer-Verlag/Wien (1999) 235–246
28. Tole, P., Pellacini, F., Walter, B., Greenberg, D.: Interactive global illumination in dynamic scenes. ACM Trans. Graph. **21** (2002) 537–546
29. Pighin, F., Lischinski, D., Salesin, D.: Progressive previewing of ray-traced images using image plane disconinuity meshing. In: Proceedings of the Eurographics Workshop on Rendering '97, London, UK, Springer-Verlag (1997) 115–125
30. Guo, B.: Progressive radiance evaluation using directional coherence maps. In: SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques, New York, NY, USA, ACM (1998) 255–266
31. Bala, K., Walter, B., Greenberg, D.P.: Combining edges and points for interactive high-quality rendering. ACM Trans. Graph. **22** (2003) 631–640
32. Sen, P., Cammarano, M., Hanrahan, P.: Shadow silhouette maps. ACM Transactions on Graphics **22** (2003) 521–526
33. Sen, P.: Silhouette maps for improved texture magnification. In: HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, New York, NY, USA, ACM (2004) 65–73
34. Bertalmio, M., Sapiro, G., Caselles, V., Ballester, C.: Image inpainting. In: SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques. (2000) 417–424
35. Masnou, S., Morel, J.M.: Level lines based disocclusion. In: Proceedings of ICIP. (1998) 259–263
36. Marvasti, F.: Nonuniform Sampling: Theory and Practice. Kluwer Academic Publishers (2001)
37. Feichtinger, H., Gröchenig, K., Strohmer, T.: Efficient numerical methods in non-uniform sampling theory. Numer. Math **69** (1995) 423–440
38. Marvasti, F., Liu, C., Adams, G.: Analysis and recovery of multidimensional signals from irregular samples using nonlinear and iterative techniques. Signal Process. **36** (1994) 13–30
39. Gu, J., Nayar, S., Grinspun, E., Belhumeur, P., Ramamoorthi, R.: Compressive structured light for recovering inhomogeneous participating media. In: ECCV. (2008)

40. Peers, P., Mahajan, D., Lamond, B., Ghosh, A., Matusik, W., Ramamoorthi, R., Debevec, P.: Compressive light transport sensing. ACM Trans. Graph. **28** (2009) 1–18
41. Sen, P., Darabi, S.: Compressive Dual Photography. Computer Graphics Forum **28** (2009) 609 – 618
42. Candès, E.J., Romberg, J., Tao, T.: Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. IEEE Trans. on Information Theory **52** (2006) 489–509
43. Donoho, D.L.: Compressed sensing. IEEE Trans. on Information Theory **52** (2006) 1289–1306
44. Rice University Compressive Sensing Resources website: (2009) http://www.dsp.ece.rice.edu/cs/.
45. Candès, E.J., Rudelson, M., Tao, T., Vershynin, R.: Error correction via linear programming. In: IEEE Symposium on Foundations of Computer Science. (2005) 295–308
46. Tropp, J.A., Gilbert, A.C.: Signal recovery from random measurements via orthogonal matching pursuit. IEEE Trans. on Information Theory **53** (2007) 4655–4666
47. Needell, D., Vershynin, R.: Uniform uncertainty principle and signal recovery via regularized orthogonal matching pursuit (2007) Preprint.
48. Wright, S., Nowak, R., Figueiredo, M.: Sparse reconstruction by separable approximation. IEEE Transactions on Signal Processing **57** (2009) 2479–2493
49. Candès, E.J., Tao, T.: Near optimal signal recovery from random projections: universal encoding strategies? IEEE Trans. on Information Theory **52** (2006) 5406–5425
50. Shannon, C.E.: Communication in the presence of noise. Proc. Institute of Radio Engineers **37** (1949) 10–21
51. Donoho, D.L., Huo, X.: Uncertainty principles and ideal atomic decomposition. IEEE Transactions on Information Theory **47** (2001) 2845–2862
52. Gonzalez, R.C., Woods, R.E.: Digital Image Processing. Addison-Wesley Longman Publishing Co., Inc. (2001)
53. LuxRender: (2009) http://www.luxrender.net/.
54. Dunbar, D., Humphreys, G.: A spatial data structure for fast Poisson-disk sample generation. ACM Trans. Graph. **25** (2006) 503–508
55. Intel Math Kernel Library: (2009) http://www.intel.com/.
56. Stanford Systems Optimization Laboratory software website: (2009) http://www.stanford.edu/group/SOL/software/lsqr.html.
57. Tsaig, Y., Donoho, D.L.: Extensions of compressed sensing. Signal Process. **86** (2006) 549–571
58. Taubman, D.S., Marcellin, M.W.: JPEG 2000: Image Compression Fundamentals, Standards and Practice. springer (2001)
59. Alper, E., Mavinkurve, S.: Image inpainting implementation (2002) http://www.eecs.harvard.edu/ sanjay/inpainting/.
60. Painter, J., Sloan, K.: Antialiased ray tracing by adaptive progressive refinement. SIGGRAPH Comput. Graph. **23** (1989) 281–288
61. Cook, R.L., Porter, T., Carpenter, L.: Distributed ray tracing. SIGGRAPH Comput. Graph. **18** (1984) 137–145
62. Hachisuka, T., Jarosz, W., Weistroffer, R.P., Dale, K., Humphreys, G., Zwicker, M., Jensen, H.W.: Multidimensional adaptive sampling and reconstruction for ray tracing. ACM Trans. Graph. **27** (2008) 1–10
63. Overbeck, R.S., Donner, C., Ramamoorthi, R.: Adaptive wavelet rendering. ACM Trans. Graph. **28** (2009) 1–12

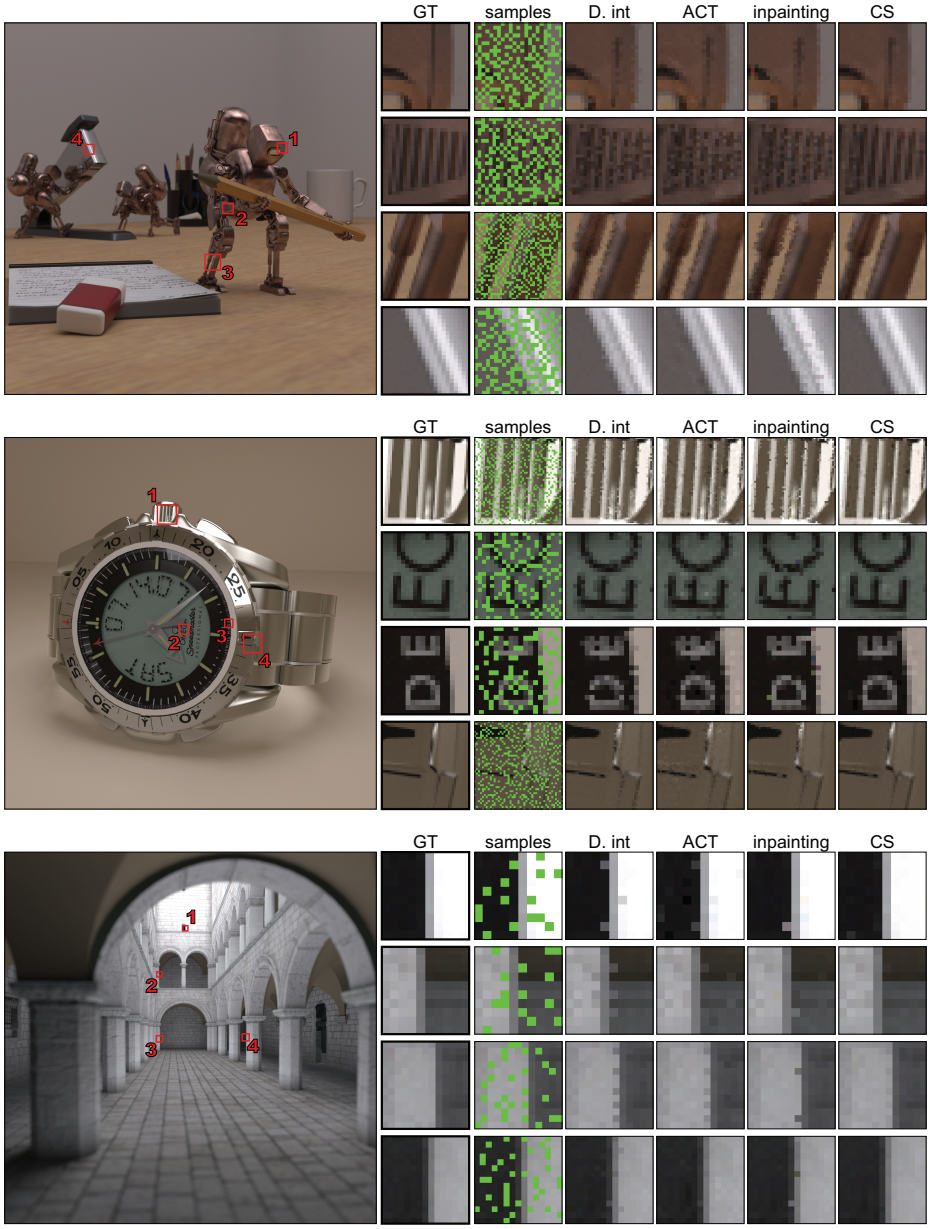


Fig. 12. Results of scenes with the reconstruction algorithms, each with a different % of computed samples. From top to bottom: ROBOTS (60%), WATCH (72%), SPONZA (87%). The large image shows the ground truth (GT) rendering to show context. The smaller columns show the ground truth (GT) of the inset region and the ray-traced pixels (unknown pixels shown in green), followed by the results of Delaunay interpolation (D. int), ACT, inpainting, and compressed sensing (CS). It can be seen that our algorithm produces higher-quality images than the others.