Eurographics Symposium on Rendering 2010
Jason Lawrence and Marc Stamminger
(Guest Editors)

Volume 29 (2010), Number 4

Compressive estimation for signal integration in rendering

Pradeep Sen and Soheil Darabi

Advanced Graphics Lab, University of New Mexico

Abstract

In rendering applications, we are often faced with the problem of computing the integral of an unknown function. Typical approaches used to estimate these integrals are often based on Monte Carlo methods that slowly converge to the correct answer after many point samples have been taken. In this work, we study this problem under the framework of compressed sensing and reach the conclusion that if the signal is sparse in a transform domain, we can evaluate the integral accurately using a small set of point samples without requiring the lengthy iterations of Monte Carlo approaches. We demonstrate the usefulness of our framework by proposing novel algorithms to address two problems in computer graphics: image antialiasing and motion blur. We show that we can use our framework to generate good results with fewer samples than is possible with traditional approaches.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Antialiasing

1. Introduction

Many algorithms in computer graphics must take definite integrals of an unknown signal, or of a signal that is known but impossible to describe in analytic form suitable for integration. We encounter this problem in a variety of application areas, from algorithms that compute the illumination in a scene to those that calculate effects such as antialiasing or motion blur for image synthesis. A common approach for solving these integrals is Monte Carlo integration, wherein a large number of random point samples of the function are used to estimate the value of the integral with some probability. As we take more samples, the variance of our estimate is reduced as $O(1/N)$ and the integral is estimated more accurately. Unfortunately, Monte Carlo methods theoretically require an infinite number of samples to fully converge. In practice, these methods require a large set of samples or we risk getting noisy results.

In this work, we study the problem of estimating and integrating unknown functions under the framework of *compressed sensing* (CS), a growing area of research in the applied mathematics community. Compressed sensing theory states that we can perfectly reconstruct a signal from a small set of linear measurements if the signal is sparse in a transform domain that is incoherent with the measurement basis. Compressed sensing is not new to computer graphics and has been applied to problems mostly in the area of light transport acquisition [GNG*08, PML*09, SD09], although a more recent application was demonstrated by the authors for rendering [SD10a]. We take a different approach than previous work by demonstrating that a better use of CS is to estimate signals for the purpose of integration during rendering. Specifically, our paper makes the following two contributions:

1. We pose the problem of calculating the integral of an unknown function for rendering applications in the framework of compressed sensing, which reveals that it is possible to accurately approximate the integral with a fixed set of point samples if the function is sparse in a transform domain.
2. We demonstrate the effectiveness of our proposed approach by demonstrating two new algorithms for antialiasing and rendering motion blur.

2. Previous Work

Researchers in the past have proposed to fit parametric models to signals before integration to reduce the variance noise of Monte Carlo integration. One example is the work of Meyer and Anderson [MA06], which fits a linear approximation to the signal using PCA to reduce noise for indirect illumination. However, traditional methods for parametric fitting such as PCA typically provide the least-squares fit to the problem, which cannot capture information with frequency larger than half the sampling rate. The compressed sensing theory upon which our work is founded, on the other hand, offers us a new way to recover some of the higher-frequency information in the signal, even beyond the sampling rate.

There is also related work that, like our own, uses sparsity in a transform domain to reduce the sampling rate or ameliorate the Monte Carlo noise. One example is the work of Bolin and Meyer [BM95], which takes a set of ray-traced samples and solves for the Discrete Cosine Transform (DCT) coefficients which fit the given data. This is another example of parametric fitting, although here they fit the signal with cosine basis functions. However, because they use least-squares to solve for these coefficients, they are limited to the range of frequencies they can recover in the result. Another example of exploiting sparsity in the transform domain to reduce Monte Carlo noise is the adaptive algorithm of Egan et al. for

motion-blur [ETH*09]. This algorithm evaluates the multidimensional signal in the transform domain to determine where to position samples. More recently, Overbeck et al. [ODR09] published an algorithm that uses wavelets for acceleration of the multidimensional integrals. Our algorithm also uses the wavelets to compress the image function for the antialiasing application, but unlike these papers it is not an adaptive algorithm in that it does not evaluate new samples based on the state of the algorithm. Rather, our rendering system simply picks the sample positions randomly, without any knowledge of the scene information, and then uses the sparsity in the transform domain to estimate the signal. These other approaches are therefore orthogonal to our own and it would be interesting future work to combine these adaptive sampling algorithms with our approach.

Finally, as mentioned in the introduction there have been recent papers in the computer graphics community that make use of compressed sensing. Most of the work focuses on applying compressed sensing to the problem of light-transport acquisition [GNG*08, PML*09, SD09]. Although these papers introduced compressed sensing to computer graphics, their algorithms do not apply to our problem because of the fundamental differences between light transport measurement and rendering. When measuring light transport, it is possible to use structured illumination to measure integral projections of fairly arbitrary functions against the signal we are trying to estimate, which makes the application of CS fairly straightforward. In our rendering application, however, this is much more difficult to do because the cost of evaluating a single pixel is not the same as that of evaluating the sum of a thousand of them (which comes for free in light transport measurement, since you can turn on one projector pixel or a thousand and measure their collective sum just as easily).

A formulation that addresses this issue was recently introduced by the authors in a method called compressive rendering [SD10a]. In that work, we used CS to accelerate rendering by only computing a fraction of the pixels in the image and then filling in the remaining ones with CS. In order to make the compressed sensing framework compatible with the random point-sampled basis, we introduced a blurred wavelet compression basis to reduce the coherence with point sampling. In this work, we use a similar framework to apply CS to the problem of integrating the image for antialiasing and motion-blur, and our approach can indeed be considered a form of compressive rendering. However, there are several fundamental differences between the two techniques.

First, the addition of our filter in the antialiasing application both increases the compatibility with the point-sampled basis as well as bandlimits the signal for downsampling. Second, we propose to estimate the signal using CS for integration rather than for direct viewing, so the slight artifacts from the reconstruction are not directly visible because we integrate over the resulting signal at the end. We find this to be a more suitable rendering application for compressed sensing. Finally, the application of our technique to motion blur demonstrates that these approaches might be better suited for rendering of animated scenes, rather than the static images rendered in the original compressive rendering work. Our results suggest that the benefit of CS for rendering applications is increased as we change the dimensionality from 2-D for a static image to 3-D for an animated sequence. This is an important observation that had not been made in previous work.

3. Compressed Sensing

Since compressed sensing (CS) is not new to computer graphics, we will only summarize the key aspects of the theory in this section. A more thorough explanation from a graphics perspective can be found in papers by Peers et al. [PML*09] and by the authors themselves [SD09, SD10a]. Readers seeking more background on CS are directed to the seminal papers in the field [CRT06, Don06].

The theory of compressed sensing allows us to reconstruct a signal from a few linear samples if it is sparse in a transform domain. Suppose there is a k-dimensional discrete signal $\mathbf{f} \in \mathbb{R}^k$ we want to estimate with a set of N random samples, a process which can be written by the linear system $\mathbf{y} = \mathbf{S}\mathbf{f}$, where $\mathbf{y}$ is the set of measured samples and $\mathbf{S}$ is an $N \times k$ sampling matrix. In our case we are doing random point-sampling, so this matrix has zeros everywhere except for single 1's in each row at the position of the random samples. If the number of samples N is a lot smaller than the signal size k, the system will be severely undetermined which makes it difficult to solve for $\mathbf{f}$ with conventional techniques.

Compressed sensing theory shows it is possible to solve for the original $\mathbf{f}$ if its transform $\hat{\mathbf{f}}$ is sparse (has very few non-zero values) in a basis Ψ, where $\hat{\mathbf{f}} = \Psi^H \mathbf{f}$. By adding the compression basis Ψ into our measurement equation, we get $\mathbf{y} = \mathbf{S}\Psi\hat{\mathbf{f}}$. Compressed sensing theory states that we can solve for $\hat{\mathbf{f}}$ uniquely [CRT06] as long as N is greater than twice the sparsity of $\hat{\mathbf{f}}$ by searching for the sparsest $\hat{\mathbf{f}}$ that solves

$$\min \|\hat{\mathbf{f}}\|_0 \quad \text{s.t. } \mathbf{y} = \mathbf{A}\hat{\mathbf{f}}, \tag{1}$$

where $\mathbf{A} = \mathbf{S}\Psi$ is a *measurement* matrix that must meet the Restricted Isometry Condition (RIC) [NV07]. The RIC effectively requires that sampling basis $\mathbf{S}$ and compression basis Ψ be as incoherent as possible. Partial Fourier matrices (randomly selected Fourier basis functions) are often used in CS research, for example, because they meet the RIC [CT06]. While we can use the simple Fourier basis for the reconstruction of a video stream in our motion-blur application, in our antialiasing application we need wavelet transforms to give us enough compression of the 2-D image. Unfortunately, the point-samples measured by the rendering system and wavelets are not incoherent, so we need to modify the wavelet basis to increase this incoherency (see Sec. 5.1).

The solution of the ℓ_0 problem in Eq. 1 unfortunately requires a brute-force combinatorial search of all $\hat{\mathbf{f}}$ vectors with sparsity less than m, which is intractable for anything other than very small problems. For this reason, the CS research community has been developing fast algorithms that find approximate solutions to the ℓ_0 problem or its ℓ_1 equivalent. In this paper, we leverage two different solvers, ROMP [NV07] (for antialiasing) and SpaRSA [WNF09] (for motion-blur).

Unfortunately, space limitations prevent us in going into detail on how these solvers work. Readers interested in implementation details should read the technical report available on the authors' website [SD10b]. The key idea is that these algorithms iteratively explore the search space of vectors $\hat{\mathbf{f}}$ that have sparse coefficients until they find a solution that approximates $\mathbf{y} = \mathbf{A}\hat{\mathbf{f}}$ with $\hat{\mathbf{f}}$ being as sparse as possible.

4. A framework for compressive integration

We now return to the problem of integrating an unknown function. Suppose signal $f(x)$ is a real-world signal (a scene, an image, etc.) which we want to integrate over some interval. Signals like this are typically compressible in a transform domain, so the logical conclusion of CS theory is that we only need a few samples to evaluate the integral accurately. We begin by projecting $f(x)$ into a set of basis functions $\psi_i(x)$, so we can reconstruct it as

$$f(x) = \sum_{i=0}^{k} a_i \psi_i(x). \tag{2}$$

If we use this basis expansion for $f(x)$, we write the integral as

$$I = \int_a^b f(x)\,dx = \int_a^b \sum_{i=0}^{k} a_i \psi_i(x)\,dx = \sum_{i=0}^{k} a_i \int_a^b \psi_i(x)\,dx, \tag{3}$$

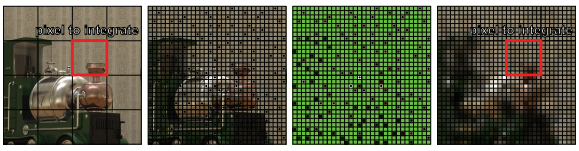
where we integrate $f(x)$ from a to b. Given that the real-world signal is sparse in domain $\psi_i(x)$, only a few of the coefficients a_i will be non-zero. We can therefore write the integral as:

$$I = a_{b_0} \int_a^b \psi_{b_0}(x)dx + a_{b_1} \int_a^b \psi_{b_1}(x)dx + \ldots + a_{b_m} \int_a^b \psi_{b_m}(x)dx, \tag{4}$$

where the $b_0 \ldots b_m$'s represent the index positions of the m non-zero coefficients. Since the basis functions ψ_i's are known and available in analytic form, their integrals are easy to pre-compute. However, we still have to determine the values of the m non-zero coefficients a_{b_i} without knowing the entire function $f(x)$. We can do this by leveraging the theory of compressed sensing, but we first have to transform our problem to the discrete domain.

We approximate the unknown function $f(x)$ with a discrete vector $\mathbf{f}$ of size k by taking uniform samples of $f(x)$. This approximation is reasonable as long as k is large enough, since it is equivalent to discretizing $f(x)$. When doing so, our uniform samples must ensure periodicity so that the sparsity of $f(x)$ in the transform domain is preserved. More details on this can be found in the associated technical report [SD10b]. Note that we do not explicitly sample $f(x)$ to create $\mathbf{f}$ (since we do not know $f(x)$ a priori), but rather we assume a k-length vector $\mathbf{f}$ exists which is the discrete version of the unknown $f(x)$ and which we will solve for through CS.

We can now take N random measurements of $\mathbf{f}$, as given by $\mathbf{y} = \mathbf{S}\mathbf{f}$ where $\mathbf{S}$ is the $N \times k$ sampling matrix, by point-sampling the original function $f(x)$ at the appropriate discrete locations. Therefore, unlike traditional Monte Carlo approaches, our random samples do not occur arbitrarily along the continuous domain of $f(x)$, 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 Eq. 1 for the coefficients that correspond to the non-zero basis functions $\hat{\mathbf{f}}$. These can then be used to evaluate our integral by Eq. 4.



(a) (b) (c) (d)

Figure 1: ***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 N samples of the signal aligned on an underlying grid of fixed resolution k. This is equivalent of taking N 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 $\mathbf{A} = \mathbf{S}\Psi$. $\mathbf{S}$ is the sampling matrix corresponding to the samples taken, Ψ is the blurred-wavelet basis described in Sec. 5.1. **(d)** Our approximation to $\mathbf{f}$, computed by applying the synthesis basis to $\hat{\mathbf{f}}$ (i.e., $\mathbf{f} = \Psi\hat{\mathbf{f}}$). We integrate this approximation over each pixel to get our antialiased result.*

Before we conclude this section, we note that computing integrals in this manner introduces two sources of bias. The first is due to the discretization of $f(x)$. Clearly, summation over a discrete function could have a slightly different value than the integral over the original signal. The second is related, but more subtle. The sampling of $f(x)$ to form the vector $\mathbf{f}$ theoretically happens at uniform intervals, and so it could be prone to aliasing since frequencies in $f(x)$ could be unbounded. Although CS can handle the reconstruction of frequencies in $\hat{\mathbf{f}}$ which are higher than the sampling rate, the higher frequencies of $f(x)$ might be irrevocably lost through aliasing when we discretized it to $\mathbf{f}$. Therefore, the $\mathbf{f}$ we are solving for using compressed sensing is not an exact representation of the original $f(x)$, which is another source of bias. We will revisit this issue again when discussing our antialiasing application in Sec. 5.2. These two sources of bias can be significantly reduced by making k as large as possible.

5. Application to antialiasing

In our first application, we use our framework for compressive integration to evaluate the integral of the continuous scene representation over the footprint of a pixel in order to perform box-filtered antialiasing. The basic idea is simple (see overview in Fig. 1). We first take a few random point samples of scene $f(x)$ per pixel. These samples are positioned on an underlying grid 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. 1 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.

As discussed in Sec. 4, bias can be introduced by the discretization of $f(x)$ in order to solve the problem with the

framework of compressed sensing. Of specific concern to this application is the aliasing that can occur when uniformly sampling $f(x)$ to get the vector $\mathbf{f}$. To avoid this, the size k of $\mathbf{f}$ must be large enough to support twice the highest frequency of $f(x)$. This is interesting, because it is different than the traditional cause of aliasing in image synthesis. Normally, aliasing is caused by insufficient sampling with respect to the highest frequency contained in the signal. In our case, however, the number of samples N does not make a difference as long as it is enough to support the *sparsity* of the signal, not the highest frequency. Rather, the *size* of the problem we are solving with CS must be large enough to accommodate all the frequencies. In most examples we tested this is not a problem, but it is easy to construct cases where $f(x)$ contains unbounded frequencies (see Fig. 7). In these cases, our algorithm will still converge quickly but there will be bias in the estimator caused by aliasing.

5.1. Suitable basis for images in point-sampled CS

Unfortunately, the Fourier basis commonly used in CS research is not suitable for our antialiasing application because images are actually not very sparse in the Fourier domain. Since the quality of our algorithm depends on the sparsity of $\hat{\mathbf{f}}$, we would like a basis that is extremely efficient at representing natural images. For this reason, we use the Daubechies-8 (DB-8) wavelet in this work. Unfortunately, this is not as easy as simply plugging in the wavelet for Ψ, since wavelets are not incoherent with the point-sampling basis $\mathbf{S}$ we use to take measurements of $\mathbf{f}$ as required by the Restricted Isometry Condition (RIC) mentioned in Sec. 3.

To overcome this, we observe that in our application we should bandlimit (antialias) the signal based on the final image resolution. Therefore, we modify the measurement equation $\mathbf{y} = \mathbf{Sf}$ to include a filter that is sized according to the final resolution, which serves to both bandlimit the signal based on the downsampling rate as well as reduce the coherency between the point-samples and our wavelet basis. Essentially, we assume that there is a blurred signal $\mathbf{f}_b$ which can be sharpened by the reversible filter Φ^{-1} to obtain $\mathbf{f}$, the signal sampled by the rendering system (in other words $\mathbf{f} = \Phi^{-1}\mathbf{f}_b$). We can then substitute this into our measurement equation as: $\mathbf{y} = \mathbf{Sf} = \mathbf{S}\Phi^{-1}\mathbf{f}_b$. Since this blurred image is also compressible in the wavelet domain, we add the wavelet compression basis to the equation to get $\mathbf{y} = \mathbf{S}\Phi^{-1}\Psi\hat{\mathbf{f}}_b$. We can now solve for $\hat{\mathbf{f}}_b$ using CS by assuming it is sparse:

$$\min \|\hat{\mathbf{f}}_b\|_0 \quad \text{s.t. } \mathbf{y} = \mathbf{A}\hat{\mathbf{f}}_\mathbf{b}, \tag{5}$$

where $\mathbf{A} = \mathbf{S}\Phi^{-1}\Psi$. In our implementation, we use a Gaussian filter for Φ. To compute the application of the filter (convolution), we observe that the filtering process is equivalent to multiplication in the frequency domain and we can write $\Phi = \mathcal{F}^H\mathbf{G}\mathcal{F}$, where $\mathcal{F}$ is the Fourier transform matrix and $\mathbf{G}$ is a diagonal matrix with values of a Gaussian function along its diagonal.

We use the amount of downsampling between our reconstructed estimate of $\mathbf{f}$ and the final antialiased image to set the variance of the Gaussian function of $\mathbf{G}$. Specifically, for a given final image resolution p (final width $\times$ height) we compute how much we will need to downsample our estimate of $\mathbf{f}$ by taking the square-root of the ratio between the size of the ROMP problem k (also width $\times$ height) and the final resolution p, or $\sqrt{k/p}$. This determines how much we must downsample $\mathbf{f}$ in one dimension to compute a pixel in the final result. We then size the spatial-domain Gaussian filter Φ so that the footprint of two standard deviations ($2\sigma_\Phi$) is slightly larger (10%) than the size of the block that will be filtered together. Therefore, we set $2\sigma_\Phi = 1.1 \times (1/2)\sqrt{k/p}$, where the division by 2 is needed because we place the Gaussian in the middle of the block.

Algorithm 1 Compressive Antialiasing algorithm

Input: size of problem k, number of samples N, and final resolution p
Output: antialiased image of target size
1: Set $\Psi \leftarrow k \times k$ DB-8 basis matrix
2: Set $\mathbf{G} \leftarrow$ diagonal matrix with a Gaussian function on the diagonal, with standard deviation $\sigma_\mathbf{G} = k/(2\pi\sigma_\Phi)$ where $\sigma_\Phi = 1.1 \times (1/4)\sqrt{k/p}$
3: Set $\Phi \leftarrow \mathcal{F}^H\mathbf{G}\mathcal{F}$, where $\mathcal{F}$ is a Fourier matrix.
4: Pick a set of random sample positions with Poisson-disk distribution aligned with grid of $\mathbf{f}$ and create sampling matrix $\mathbf{S}$
5: Set $\mathbf{A} \leftarrow \mathbf{S}\Phi^{-1}\Psi$ and $\mathbf{A}^\dagger \leftarrow \Psi^{-1}\Phi\mathbf{S}^T$
6: Use ray tracer to render samples $\mathbf{y}$ at selected sample positions $\mathbf{S}$
7: Run ROMP with measured samples $\mathbf{y}$, matrices $\mathbf{A}$ and $\mathbf{A}^\dagger$, to get $\hat{\mathbf{f}}_b$
8: Compute $\mathbf{f}_b \leftarrow \Psi\hat{\mathbf{f}}_b$ which is the estimated high-resolution filtered image and integrate over each pixel to get AA version of size p
9: **return** final image

For example, suppose ROMP is run on a problem of size $k = 1024 \times 1024$ so that we reconstruct a discrete approximation $\mathbf{f}$ of the continuous image at 1024×1024. If our goal is to produce an antialiased image of size 256×256 (like many of the results in this paper), we will average down 4×4 values of this reconstruction to compute a single pixel of the final image. If we place the Gaussian kernel at the center of the pixel sample, we want the $2\sigma_\Phi$ boundary to extend just a little over $(1/2)\sqrt{1024^2/256^2} = 2$, so we set it 10% more at $2\sigma_\Phi = 2.2$. To set the variance for the frequency-domain Gaussian $\mathbf{G}$, we recall the relationship between the standard deviations in the spatial and the frequency domain: $\sigma_\mathbf{G} = k/(2\pi\sigma_\Phi)$. Therefore, given the size of the ROMP problem we are solving k and our target image size p, it is straightforward to compute the variance of the Gaussian $\sigma^2_\mathbf{G}$.

We must also compute the inverse of the filter Φ^{-1} since $\mathbf{A} = \mathbf{S}\Phi^{-1}\Psi$. Here, $\Phi^{-1} = \mathcal{F}^H\mathbf{G}^{-1}\mathcal{F}$, where $\mathbf{G}^{-1}$ is also a diagonal matrix that should have values $\mathbf{G}^{-1}_{i,i}$ along its diagonal. We must be careful when inverting the Gaussian function because of noise amplification, so we use a linear Wiener filter to do this [SD10a]. This means that the diagonal elements of $\mathbf{G}^{-1}$ have the form $\mathbf{G}^{-1}_{i,i} = \mathbf{G}_{i,i}/(\mathbf{G}^2_{i,i} + \lambda)$. In our experiments, we use $\lambda = 0.3$. Since the ROMP algorithm also requires a pseudoinverse matrix $\mathbf{A}^\dagger$, we set $\mathbf{A}^\dagger = \Psi^{-1}\Phi\mathbf{S}^T = \Psi^{-1}\mathcal{F}^H\mathbf{G}\mathcal{F}\mathbf{S}^T$. We can now summarize our entire antialiasing algorithm. First, we take N random samples of the scene with a ray tracer to form our measurement vector $\mathbf{y}$, and then use ROMP to solve for $\hat{\mathbf{f}}_b$, the wavelet transform of the estimate of the high-resolution image which has been adequately filtered for downsampling. We then take the inverse wavelet transform to compute $\mathbf{f}_b = \Psi\hat{\mathbf{f}}_b$. Once

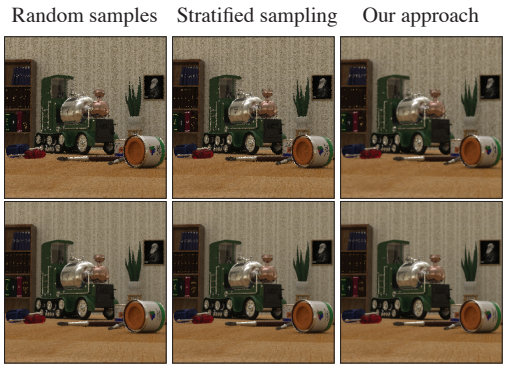


Figure 2: ***Visual comparison for*** **TRAIN** ***scene.*** *The top row has 1 sample/pixel, the bottom has 4.*

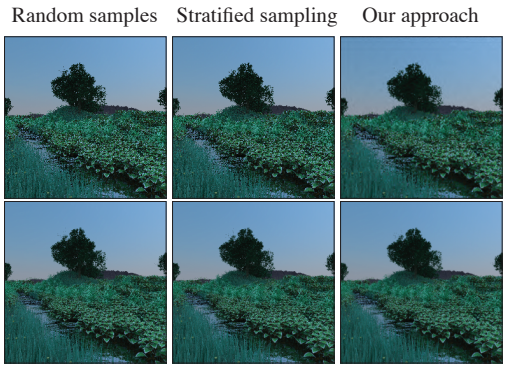


Figure 3: ***Visual comparison for*** **GARDEN** ***scene.*** *The top row has 1 sample/pixel, the bottom has 4.*

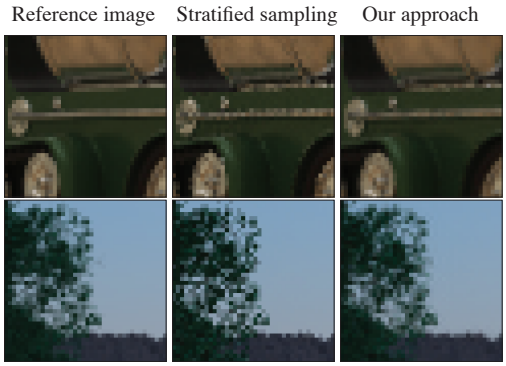


Figure 4: ***Comparison to reference image.*** *Insets from the* TRAIN *image (top) and the* GARDEN *image (bottom) compare traditional stratified sampling and our approach at 4 samples/pixel to a reference image, rendered at 16 stratified samples/pixel.*

we have our filtered image in the spatial domain, we downsample it with a box filter to compute our final image. Pseudocode for our entire algorithm is listed in Alg. 1.

The addition of the blurring filter Φ means that our measurement matrix $\mathbf{A}$ is now composed of two parts: the point-samples $\mathbf{S}$ and the blurred wavelet matrix $\Phi^{-1}\Psi$ which acts as our compression basis. This is related to the our earlier approach [SD10a], where we used a similar formulation to reduce the overall coherence between the point-samples and the compression basis. However, there are a two subtle, but important, differences between the approaches. First, in this work we size our Gaussians based on the size of the final image, while earlier we did so empirically based on the sampling rate. This new approach is better for antialiasing since it enables proper filtering of the signal before downsampling. The second difference is that while our original work applies an inverse filter to $\mathbf{f}_b$ to get a sharp, high resolution result for direct viewing, we simply use the filtered result $\mathbf{f}_b$ for downsampling because it has been appropriately bandlimited. This gives us better results than the previous approach.

5.2. Results of antialiasing algorithm

To test our algorithm, we integrate it into two Monte Carlo rendering systems: PBRT [PH04] and LuxRender [Lux]. This process is straightforward, since we simply modify the code to render the image at the specified sample positions and then feed the measured values $\mathbf{y}$ to the ROMP solver, which we implemented in C based on the MATLAB code provided by the ROMP authors [NV07]. The samples selected from the underlying grid are distributed in a Poisson-disk distribution to provide better reconstruction.

To evaluate our results, we compare against a traditional antialiasing technique using random samples with Monte Carlo integration and another that uses jittered, stratified sampling to reduce variance. We run these three algorithms on several test scenes: three simple ones (TRIANGLE, BUDDHA, and CHECKERBOARD) and two more sophisticated ones (TRAIN and GARDEN) to demonstrate that we can handle complex textures and geometry. The images for TRAIN and GARDEN are rendered at 256×256 and the others at 128×128 in order to highlight the antialiasing process.

First, we examine the visual quality of our technique on the two scenes rendered with LuxRender. Figs. 2 and 3 compare the three different algorithms using a different number of samples per pixel for each row. We can see that even with one sample per pixel, our algorithm begins to produce softer edges, something that is not possible with the other techniques. When we have 4 samples per pixel, we get a result which is comparable to the 9-sample/pixel result for stratified sampling. To see this effect more clearly, we direct the reader's attention to Fig. 4 which shows a comparison of stratified and our approach to a reference image rendered with 16 samples per pixel for insets in the two images. The visual quality can also be appreciated when we zoom out slightly from the images (on the pdf). The images from random and stratified Monte Carlo integration show more jagginess than those of our approach.

Next, we study the effectiveness of our technique in reducing the variance of the estimated pixel values after integration. Fig. 5 shows the variance curves for three different pixels in two simple scenes, each rendered with PBRT at 128×128. To make these plots, we repeated the experiment 50 times for each sampling rate and computed the variance of a grayscale version of the RGB pixel. Our algorithm results in a substantial reduction in variance for these pixels,

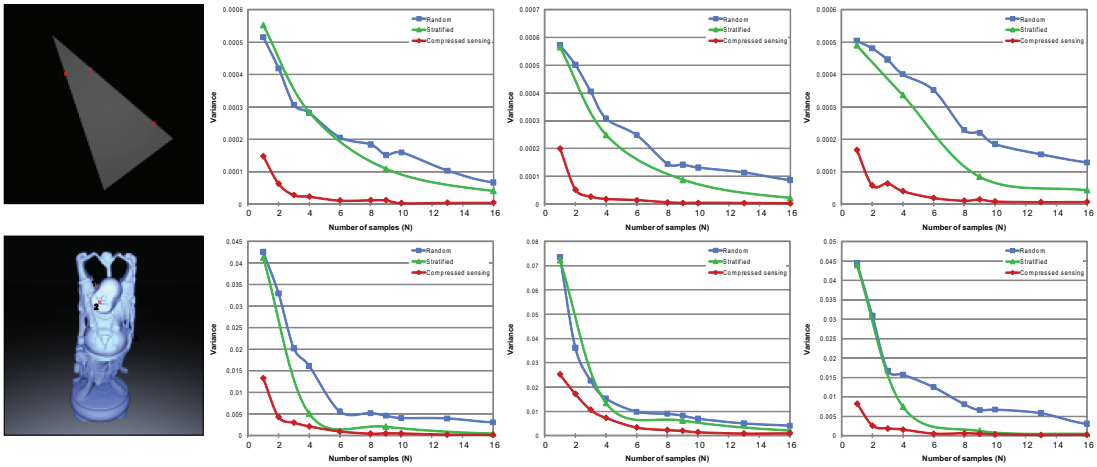


Figure 5: ***Variance vs. number of samples.*** *For two simple scenes (*TRIANGLE *and* BUDDHA*), we plot the per-pixel variance as a function of the number of samples for all three antialiasing algorithms for the three selected pixels shown.*

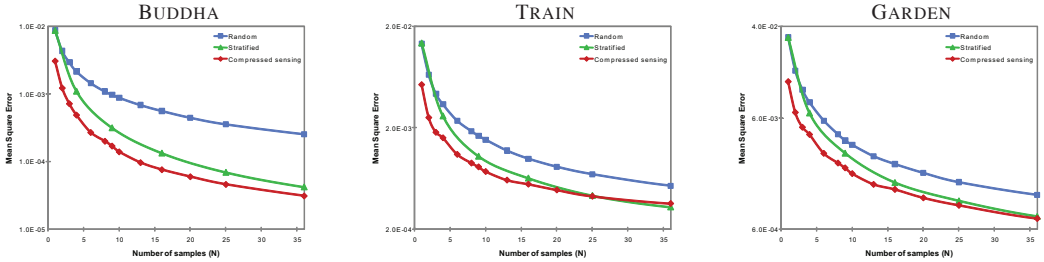


Figure 6: ***Mean Square Error vs. the number of samples.*** *We compare the MSE of the antialiased image with the ground truth AA (*16×16 *stratified samples) for the three algorithms for four of the scenes. Note that these are log plots.*

something we observed for most pixels in these images (with exception of the constant-colored pixels, of course).

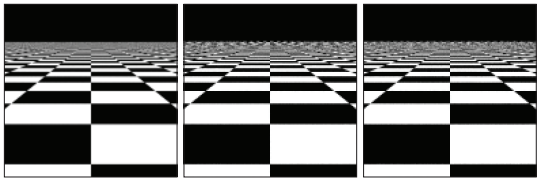
Although our algorithm is biased, the bias is small as seen in the mean-squared error (MSE) curves of each technique compared to a reference image antialiased using stratified sampling with 256 samples (see Fig. 6). The MSE is computed for the entire image, not just the edge pixels, so the curves are a little closer than they would be if we only compared the regions that needed antialiasing. Nevertheless, we can see that our algorithm has an overall lower MSE than the other techniques, especially considering that this is a log plot. Therefore, our algorithm not only converges faster, but its answer is also closer to that of the reference rendering.

In terms of run-time, our unoptimized ROMP reconstruction takes about 130 seconds to compute for the two images for Figs. 2 and 3 when we reconstruct the signal for $k = 1024^2$ with one sample per pixel using a desktop with an Intel Core i7 860 2.8 GHz CPU with 4 GB RAM. For comparison, the 4-sample per pixel TRAIN scene takes 20.7 minutes to render at 256×256 and the GARDEN scene 37.6 minutes. The reference images used for the MSE calculations were rendered at 128×128 at 256 samples/pixel and took approximately 5 hours and 10 hours respectively for these two scenes. In terms of memory consumption, we avoid storing the $\mathbf{A}$ matrix in memory for the ROMP algorithm by using a functional representation to describe $\mathbf{A}$ in the solver, which has dimensions $N \times k$ and can be quite large for normal image sizes. This makes the memory overhead of our algorithm negligible and it is dominated by the memory required for the rendering system.

Finally, we note that the aliasing that can occur by discretizing the function $f(x)$ to solve it with ROMP is independent of the number of samples measured N, so we can increase the size of the problem k without changing the sampling rate in order to ameliorate this effect. Fig. 7 shows the result of applying our algorithm on the CHECKERBOARD scene. If we use a ROMP reconstruction of 1024×1024 to produce a 128×128 pixel image, we get some aliasing near the horizon because of the discretization of $\mathbf{f}$. However, if we increase the size of the problem we are solving to 2048×2048 but keep the number of samples *exactly* the same (36 samples/pixel), the aliasing is substantially reduced. The only thing that changes when we do this is the size of the matrix $\mathbf{A}$ and vector $\hat{\mathbf{f}}$ in our ROMP formulation. Of course, the time to solve the ROMP problem also increases by $n \log n$, as shown in our earlier work [SD10a]. It is interesting that the aliasing is not dependent on the sampling rate but rather on the resolution of the underlying conceptual problem we are trying to solve.

6. Application to Motion Blur

We now describe the application of our framework to a second problem: the rendering of motion blur. Motion blur oc-



(a) (b) (c)

Figure 7: ***Problem with aliasing when discretizing the signal.*** *In the* CHECKERBOARD *scene, the frequencies near the horizon are so high that we introduce aliasing when discretizing $f(x)$ in order to solve it with CS there is aliasing.* ***(a)*** *Ground-truth AA image computed with $2,500$ stratified samples per pixel.* ***(b)*** *Image produced by our compressive antialiasing technique, using 36 samples per pixel and solving the CS problem with $k = 1024^2$. Although the pixel variance is greatly reduced, the bias results in visible aliasing artifacts near the horizon.* ***(c)*** *If we simply increase the size of the problem to $k = 2048^2$ but keep the number of samples the same, we see that the aliasing is substantially reduced.*

Algorithm 2 Compressive Motion Blur Algorithm

Input: size of problem k (k = width $\times$ height $\times$ length of spatio-temporal volume $\mathbf{f}$), number of samples N
Output: motion-blurred image
1: Set $\Psi \leftarrow k \times k$ 3-D Fourier basis matrix
2: Pick a set of N random sample positions aligned with grid of $\mathbf{f}$ and create sampling matrix $\mathbf{S}$
3: Set $\mathbf{A} \leftarrow \mathbf{S}\Psi$ and $\mathbf{A}^\dagger \leftarrow \Psi^{-1}\mathbf{S}^T$
4: Use ray tracer to render samples $\mathbf{y}$ at selected sample positions $\mathbf{S}$
5: Run SpaRSA with measured samples $\mathbf{y}$, matrices $\mathbf{A}$ and $\mathbf{A}^\dagger$ to get $\hat{\mathbf{f}}$
6: Compute $\mathbf{f} \leftarrow \Psi\hat{\mathbf{f}}$ and integrate over time to get the motion-blurred result
7: **return** motion blurred image

curs 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 [CPC84]. Recently, a few adaptive algorithms have been proposed (e.g., Hachisuka et al. [HJW*08] and Overbeck et al. [ODR09]) that can render motion blur effects. Since these techniques are considered state-of-the-art techniques in rendering, we compare our approach against them in this section.

Conceptually, our approach is very similar to that of our antialiasing algorithm. We first take a set of samples of 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 transform domain Ψ. 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.

However, there are two important differences from our antialiasing algorithm. First, recall that the reconstruction of the static image for antialiasing required the addition of a filter to the wavelet basis to make it compatible with point-sampling. Since we are now dealing with a volume of pixel data, the correlation between pixels has been increased by an extra dimension (the image data is now correlated not only in x and y but also in t). We have found experimentally in the scenes

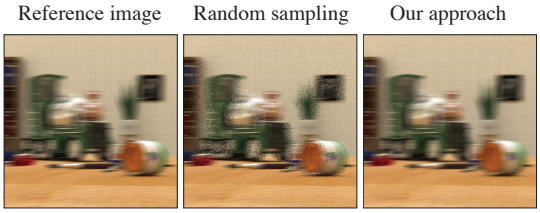


Figure 8: ***Visual comparison of motion blur results for the*** TRAIN ***scene.*** *The reference image was rendered with 70 temporal samples/pixel, while the other two where rendered with a single random sample per pixel in time. Our result was reconstructed assuming a spatio-temporal volume of 24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

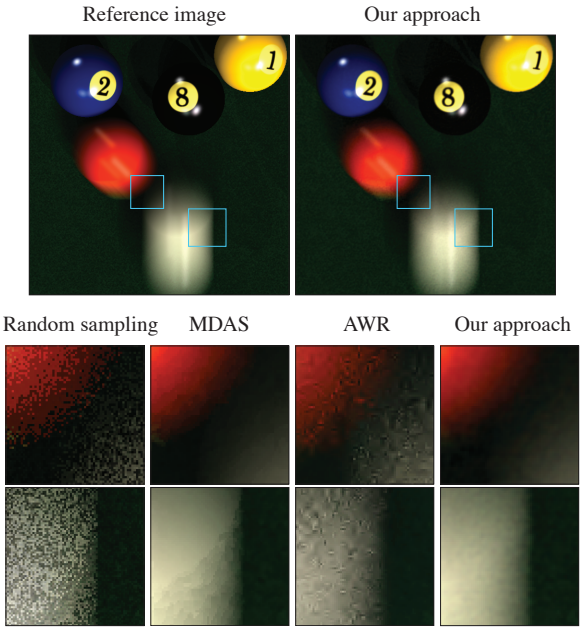


Figure 9: ***Comparison with other algorithms.*** *Here we render the* BILLIARDS *scene using our approach, as well as the MDAS [HJW*08] and adaptive wavelet rendering (AWR) [ODR09] algorithms, all at 4 samples per pixel. The reference image was rendered with 128 samples/pixel. Images were rendered at a resolution of* 500×500.

that we tested that the sparsity of the signal in the Fourier domain is now sufficient to allow for adequate reconstruction by CS algorithms from a small set of point samples. This means that we can use the Fourier basis for Ψ, which is compatible with point-sampling and so we do not need the more complex formulation we introduced in Sec. 5.1 for this application. Second, because we are dealing with a simple Fourier basis, we found that the SpaRSA solver worked more robustly and faster for reconstructing the video data sets. Readers looking for the implementation details for SpaRSA are referred to the associated technical report [SD10b]. This report also contains implementation details for the overall algorithm to achieve the results shown in the paper. A summary of the overall algorithm is given in Alg. 2.

First, we test the motion blur on the train scene in Fig. 8 by

moving the camera. We see that even with one sample/pixel we get a reasonable motion blur for this image, especially when compared to the simple Monte Carlo approach. To test against more sophisticated algorithms, we run our algorithm on the BILLIARDS scene used in previous papers and compare its results against traditional Monte Carlo, the multidimensional adaptive sampling (MDAS) method of Hachisuka et al. [HJW*08], and Overbeck et al.'s adaptive wavelet rendering (AWR), all at 4 samples per pixel (see Fig. 9). For MDAS and AWR we used the code provided by the respective authors. We notice that both MDAS and AWR have strong artifacts in the motion blur regions as shown by the insets. The MDAS has piecewise constant regions (due to the nearest-neighbor interpolation they use in the time domain) while the AWR has noticeable wavelet artifacts. Although our approach does have some artifacts because of the Fourier compression, these are much more reduced than those of the two other approaches.

6.1. Extensions to rendering of animated scenes

One interesting observation is that the quality of the reconstructed frames of the spatio-temporal volume is actually quite high. This suggests compressed sensing could also be used to render animated scenes using our approach, which is an important difference from our earlier work [SD10a]. This is also a significant advantage over the more complex adaptive approaches that have been proposed (e.g., MDAS [HJW*08] or AWR [ODR09]) 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, so it fully computes the entire spatio-temporal volume which we can view as frames in the video sequence.

To demonstrate this, we show individual frames in Fig. 10 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 [SD10a], which reconstructs the set of static images individually using sparsity in the wavelet domain. The second column of Fig. 10 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 previsualization for high-end rendering, since we get a reasonable pre-viz quality image with 100× less samples. The reconstruction time is less than a minute per frame using the unoptimized MATLAB SpaRSA implementation available from the authors website [WNF09], 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. 10 (top-right image) in less than 2 hours.

It is worth noting that we also experimented with reconstructing images with a Fourier basis in our earlier work on compressive rendering [SD10a], but obtained poor results. It seems that increasing the dimensionality of the problem (from 2-D to 3-D) in this work has improved the sparsity of these scenes in the Fourier domain enough to have successful reconstructions using the Fourier basis. This suggests that perhaps higher dimensional problems might compress even better and work better with this framework.

7. Limitations and future work

One limitation of our approach is that we assume that the signals we are integrating are sparse. Real-world signals are typically compressible in a transform domain but not actually sparse, and therefore we cannot guarantee the exact recovery of the integral since the CS theory is applicable only to sparse signals. For example, we will fail to compute the integral in a pixel for antialiasing if our samples happen to miss a small white object in a black background, since this signal is not sparse in either the Fourier or the blurred wavelet domain. However, these artifacts tend to be small; the smaller the object the more likely we are to miss it with our samples but also the less contribution it will have to the over all integral. So overall there is benefit to using our approach.

This work also raises interesting directions for future work. In the short term, we need to study better basis functions Ψ that compress the pixel data while maintaining incoherency with the point samples for both static images as well as video sequences. In addition, although we focused on antialiasing and motion blur in this paper, there are many other situations in computer graphics where we want to integrate over a sparse function, such as illumination and reflection. Our framework could be used to significantly improve the quality of these other algorithms as well.

8. Conclusion

In this paper, we have proposed a novel framework for integrating unknown functions based on the theory of compressed sensing. By assuming that the signal is sparse in a transform domain, the framework can solve for the most significant transform coefficients and use them to compute the integral. We demonstrate two practical examples of our approach by proposing new algorithms for antialiasing and motion blur for ray tracing renderers. In the antialiasing application, our results are better than either random or stratified jittered sampling, common approaches for antialiasing. For motion blur, we have less artifacts at a given sampling rate than state-of-the-art approaches. This is just the beginning of

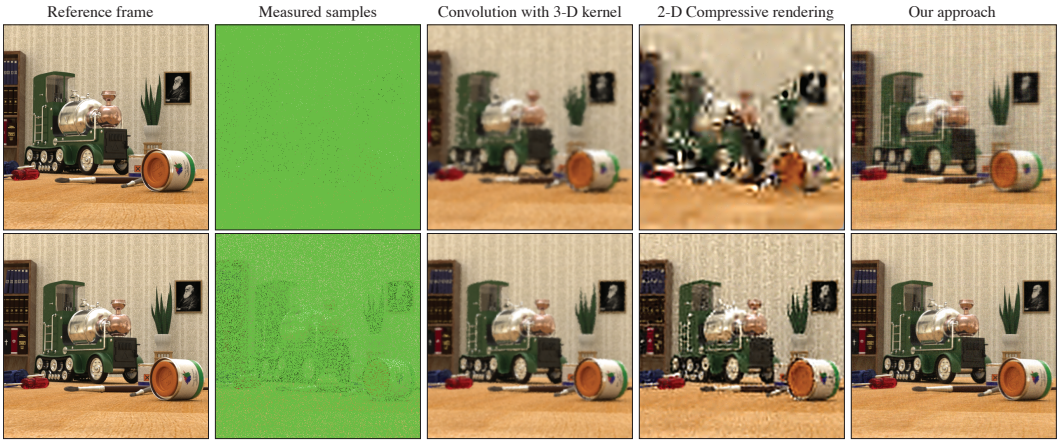


Figure 10: ***Reconstructing frames in an animated sequence.*** *We apply our approach to the rendering of individual frames in an animated sequence by leveraging the coherence both spatially and temporally in the Fourier domain. The rows represent different frames in the* TRAIN *sequence with 1% sampling rate for the top row, 10% for the bottom. The first column shows the reference frame of the fully-rendered sequence, the second column shows the samples available for this particular frame (unknown samples shown in green), the third 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 fourth column shows the results of the author's earlier compressive rendering algorithm which only uses 2-D static images to reconstruct the result, and the last column is our approach. These images are rendered at* 512×512 *with 128 frames in the sequence.*

the exploration of this research area, which seems to have a lot of potential applications in computer graphics.

Acknowledgments The authors would like to thank Lei Xiao for putting together the results in the motion blur section. Maziar Yaesoubi, Nima Khademi Kalantari, and Vahid Noormofidi also helped to acquire some of the results presented. The video for this paper was prepared by Mauricio Gómez. We also thank Toshiya Hachisuka and Ryan Overbeck for providing us with the source code for their respective implementations. The BILLIARDS scene was provided by T. Hachisuka, the TRAIN scene by J. Peter Lloyd, and the GARDEN scene is from the PBRT text [PH04]. This work was made possible by the NSF CAREER Award #0845396 "A Framework for Sparse Signal Reconstruction for Computer Graphics."

References

[BM95] 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), pp. 409–418. 1

[CPC84] COOK R. L., PORTER T., CARPENTER L.: Distributed ray tracing. *SIGGRAPH Comput. Graph. 18*, 3 (1984), 137–145. 7

[CRT06] CANDÈS E. J., ROMBERG J., TAO T.: Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *IEEE Trans. on Information Theory 52*, 2 (Feb. 2006), 489–509. 2

[CT06] CANDÈS E. J., TAO T.: Near optimal signal recovery from random projections: universal encoding strategies? *IEEE Trans. on Information Theory 52*, 12 (Dec. 2006), 5406–5425. 2

[Don06] DONOHO D. L.: Compressed sensing. *IEEE Trans. on Information Theory 52*, 4 (Apr. 2006), 1289–1306. 2

[ETH*09] EGAN K., TSENG Y.-T., HOLZSCHUCH N., DURAND F., RAMAMOORTHI R.: Frequency analysis and sheared reconstruction for rendering motion blur. *ACM Trans. Graph. 28*, 3 (2009), 1–13. 2

[GNG*08] GU J., NAYAR S., GRINSPUN E., BELHUMEUR P., RAMAMOORTHI R.: Compressive structured light for recovering inhomogeneous participating media. In *ECCV* (Oct 2008). 1, 2

[HJW*08] 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*, 3 (2008), 1–10. 7, 8

[Lux] LuxRender. http://www.luxrender.net/. 5

[MA06] MEYER M., ANDERSON J.: Statistical acceleration for animated global illumination. *ACM Trans. Graph. 25*, 3 (2006), 1075–1080. 1

[NV07] NEEDELL D., VERSHYNIN R.: Uniform uncertainty principle and signal recovery via regularized orthogonal matching pursuit, 2007. Preprint. 2, 5

[ODR09] OVERBECK R. S., DONNER C., RAMAMOORTHI R.: Adaptive wavelet rendering. *ACM Trans. Graph. 28*, 5 (2009), 1–12. 2, 7, 8

[PH04] PHARR M., HUMPHREYS G.: *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004. 5, 9

[PML*09] PEERS P., MAHAJAN D., LAMOND B., GHOSH A., MATUSIK W., RAMAMOORTHI R., DEBEVEC P.: Compressive light transport sensing. *ACM Trans. Graph. 28*, 1 (2009), 1–18. 1, 2

[SD09] SEN P., DARABI S.: Compressive Dual Photography. *Computer Graphics Forum 28*, 2 (2009), 609 – 618. 1, 2

[SD10a] SEN P., DARABI S.: Compressive rendering: A rendering application of compressed sensing. *IEEE Transactions on Visualization and Computer Graphics* (2010), accepted. 1, 2, 4, 5, 6, 8

[SD10b] SEN P., DARABI S.: *Details and Implementation for Compressive estimation for signal integration in rendering*. Tech. Rep. EECE-TR-10-0003, University of New Mexico, 2010. 3, 7

[WNF09] WRIGHT S., NOWAK R., FIGUEIREDO M.: Sparse reconstruction by separable approximation. *IEEE Transactions on Signal Processing 57*, 7 (July 2009), 2479–2493. 2, 8