Compressive Rendering: A Rendering Application of Compressed Sensing

Pradeep Sen, *Member, IEEE,* and Soheil Darabi, *Student Member, IEEE*

Abstract—Recently, there has been growing interest in compressed sensing (CS), the new theory that shows how a small set of linear measurements can be used to reconstruct a signal if it is sparse in a transform domain. Although CS has been applied to many problems in other fields, in computer graphics it has only been used so far to accelerate the acquisition of light transport. In this paper, we propose a novel application of compressed sensing by using it to accelerate ray-traced rendering in a manner that exploits the sparsity of the final image in the wavelet basis. To do this, we raytrace only a subset of the pixel samples in the spatial domain and use a simple, greedy CS-based algorithm to estimate the wavelet transform of the image during rendering. Since the energy of the image is concentrated more compactly in the wavelet domain, less samples are required for a result of given quality than with conventional spatial-domain rendering. By taking the inverse wavelet transform of the result, we compute an accurate reconstruction of the desired final image. Our results show that our framework can achieve high-quality images with approximately 75% of the pixel samples using a *non-adaptive* sampling scheme. In addition, we also perform better than other algorithms that might be used to fill in the missing pixel data, such as interpolation or inpainting. Furthermore, since the algorithm works in image space, it is completely independent of scene complexity.

Index Terms—Rendering, ray tracing, sampling and reconstruction, compressed sensing.

1 Introduction

One of the fundamental problems in computer graphics is *rendering*, the process of synthesizing an image from scene information. Although the final, rendered image can be efficiently compressed with transform-coding compression algorithms, most rendering systems expend their effort in rendering every single pixel in the spatial domain of the image first, only to then discard the redundant information through either static (e.g., JPEG or JPEG2000) or video (e.g., MPEG or MPEG4) image compression. This raises the question that inspired this work: is there a way to exploit the sparsity in a transform domain directly during rendering so that we render only the "important" information in the image, as opposed to rendering everything first and then throwing away most of the useless information?

This question seems to imply that we would need to do the rendering directly in the transform domain. Unfortunately, taking an arbitrary transform of a scene is extremely difficult for anything other than simple, analytic scene representations. In addition, most of the elegant rendering algorithms to date (e.g., ray tracing [1], REYES [2], and scanline algorithms [3]) do not map to other domains. Therefore, a framework that exploits sparsity in a transform domain has to be compatible with the traditional point-sampling methods common to all of these rendering algorithms.

Although other researchers have studied this problem using conventional signal processing techniques (e.g., the Bolin and Meyer work [4] we discuss in Sec. 2.1), in this paper we focus on the general sampling-and-reconstruction problem and suggest a novel approach for reconstructing the rendered image in the transform domain by leveraging recent results from the field of compressed sensing (CS). The basic idea is simple: we render only a fraction of the pixels in the image then try to estimate the values of the missing pixels to complete the rest of the image. Since this is clearly an ill-posed problem, we use compressed sensing as a regularization technique to estimate these unknown values. In other words, by assuming that the final image is compressible in a transform domain, we can use the optimization algorithms developed by the CS community to approximate it accurately using a smaller number of spatial samples than with conventional approaches. We introduce the term *compressive rendering* (or *compressed* rendering) to describe algorithms that exploit sparsity in a transform domain to make rendering more efficient in this manner.

To implement our framework, we first use a ray tracer [1] to compute a fixed subset of the pixels in the final image. We note that our sampling algorithm is not adaptive since we never raytrace new samples based on the latest image reconstruction, but rather we randomly select a set of pixels to be computed without knowing anything about the image. These measured pixels serve as our random linear projection of the image "signal," which is then assumed to be sparse in a transform domain so that we can use CS to reconstruct the missing samples. Our experiments indicate that this simple approach produces better images than other methods for filling in the missing data. In addition, since we focus on the fundamental sample-reconstruction problem and operate in image space, our approach is independent

• *P. Sen and S. Darabi are with the UNM Advanced Graphics Lab in the Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque, NM, 87131. E-mail: psen at ece.unm.edu*

of traditional ray-tracing acceleration approaches [5]. In other words, we do not accelerate the tracing of the rays or the computation of the samples, but rather focus on the problem of generating a better image with the given samples. Specifically, this paper makes three contributions to the field of rendering:

1) We apply the theory of compressed sensing to the problem of rendering. Although CS has been used in computer graphics to accelerate light transport acquisition [6], [7], [8], it has not yet been applied to rendering.
2) We present a novel approach for working with wavelets within the framework of compressed sensing. As described in Sec. 3.2, CS requires incoherence (see Eq. 4) between the sampling basis and the compression basis. In our case, rendering systems perform point sampling while image compression is best done with wavelets, which unfortunately are not incoherent bases. We propose a way to overcome this challenge and render images efficiently using wavelets (Sec. 4.1).
3) Leveraging the first two contributions, we develop a novel framework for compressive rendering that synthesizes high-quality images at lower sampling rates using a simple, non-adaptive algorithm (Sec. 4.2).

2 Related Work

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., [11], [12]) for more detail. For background on wavelets, the texts by Stollnitz et al. [13] and Mallat [14] 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. [15] 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., [16], [17], [18].

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 [19], [20] or Haar wavelets [21], [22]. 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 [23], [24] and wavelet domains [25] 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 [4]. 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. Therefore, by posing the problem of determining the value of missing pixels 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 [5]. As we mention in the introduction, these methods are orthogonal to ours since we do not accelerate the ray tracing process 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 [1], [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 this work we use the CDF 9/7 wavelet transform because it has been shown to be very good at compressing imagery. It is unclear how existing adaptive methods could be modified to use this basis. Furthermore, although our method uses a fixed, non-adaptive sampling pattern to reconstruct the image, it can also be extended to an adaptive algorithm that places new samples based on the most recent reconstruction of the image. We show results of this in Sec. 6.2.

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].

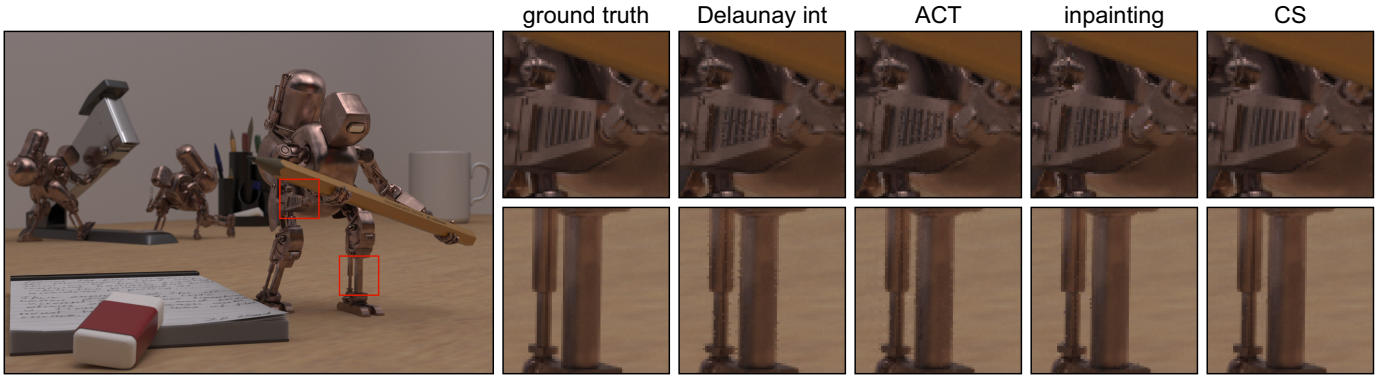


Fig. 1. Our framework exploits the sparsity of the image in the wavelet domain to accelerate ray-traced rendering. In this figure, we render our ROBOTS test scene with 80% of the total pixel samples using a fixed, non-adaptive sampling pattern. We then reconstruct the missing pixels using various algorithms. The large image on the left is cropped from the result of our compressive rendering framework, which is comparable in quality to the original image. For comparison, the leftmost inset shows a magnified portion of the ground truth, fully-rendered image. We then show the insets reconstructed from the 80% samples using interpolation, ACT [9] and inpainting [10], and finally our approach using compressed sensing to reconstruct the image using a CDF 9/7 wavelet for compression. Our result is the highest in quality and comparable to the fully-rendered ground truth, despite the fact that it took over an hour less to render.

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 image in a transform domain in order to reconstruct it. This allows us to faithfully reconstruct edges in the image, as can be seen by our results in Fig. 1.

Perhaps our work is most closely related to those approaches that fundamentally improve the process of sampling/reconstruction, such as the work by Mitchell [26]. One of the more similar papers in recent years is the work of Hachisuka et al. [34] which presents a novel sampling/reconstruction technique for improving rendering. Like our approach, theirs generates better quality images in shorter times than with standard techniques. However, while they focus on sampling the multidimensional scene information efficiently, we only deal with sampling the 2-D image plane and instead focus on reconstructing the image efficiently in a transform domain. It would be interesting to combine the two approaches in future research.

2.3 Reconstruction of missing data

Our approach to rendering only computes a fraction of the pixels in the image and uses compressed sensing to "guess" the values of the unrendered pixels. In computer graphics and vision, there are techniques such as inpainting [10] and hole-filling [35] that are also used to fill in missing pixel data. 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 rendering application, the random nature of the rendered pixels makes our application fundamentally different from that of typical inpainting, 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. 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 pixel samples 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 [9] 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 [4] described earlier. Another approach, known as the Marvasti method [37], 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 com-

ponents 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 later in the paper.

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 three published works that have used CS, all in the area of light-transport acquisition [6], [7], [8]. 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 n pixels is not as easy as calculating the value of a single pixel, it is less obvious how to apply the traditional CS approaches to the rendering problem.

3 Compressed Sensing Theory

The area of compressed sensing is relatively new to computer graphics. For this reason, we summarize some of its key theoretical results in this section to make our approach understandable to our community. 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. [38] and Donoho [39] as well as the extensive CS literature available through the Rice University online repository [40] 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 $\mathbf{x} \in \mathbb{R}^n$ we are trying to estimate with k random samples, where $k \ll n$. We can write this sampling process with the linear sampling equation $\mathbf{y} = \mathbf{S}\mathbf{x}$, 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 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 ($\mathbf{x}$) is going to be estimated from only k pixel samples ($\mathbf{y}$).

Initially, it seems that perfect estimation of $\mathbf{x}$ 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 $\mathbf{x}$ is sparse in some transform domain $\hat{\mathbf{x}} = \mathbf{\Psi}\mathbf{x}$. Mathematically, the signal $\hat{\mathbf{x}}$ 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): $\|\hat{\mathbf{x}}\|_0 \leq m$. This 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 $\hat{\mathbf{x}}$ into our sampling equation:

n	size of final image in pixels
k	number of pixels sampled by ray tracer
m	number of non-zero coefficients in transform domain
$\mathbf{x}$	high-resolution final image, represented by an $n \times 1$ vector
$\hat{\mathbf{x}}$	image transform, represented by a $n \times 1$, m-sparse vector
$\mathbf{y}$	$k \times 1$ vector of pixel samples of $\mathbf{x}$ generated by ray tracer
$\mathbf{S}$	$k \times n$ sampling matrix of ray tracer, s.t. $\tilde{\mathbf{x}} = \mathbf{S}\mathbf{x}$
$\mathbf{\Psi}$	$n \times n$ linear transform "compression" matrix with inverse $\mathbf{\Psi}^{-1}$
$\mathcal{F}$	$n \times n$ orthogonal Fourier transform matrix
$\mathbf{G}$	$n \times n$ diagonal matrix with values of a Gaussian function
$\mathbf{\Phi}$	$n \times n$ invertible blur filter, $\mathbf{\Phi} = \mathcal{F}^H \mathbf{G} \mathcal{F}$
$\mathbf{A}$	$k \times n$ "measurement" matrix, $\mathbf{A} = \mathbf{S}\mathbf{\Psi}^{-1}$

TABLE 1. Notation used in this paper.

$$\mathbf{y} = \mathbf{S}\mathbf{x} = \mathbf{S}\mathbf{\Psi}^{-1}\hat{\mathbf{x}} = \mathbf{A}\hat{\mathbf{x}}, \tag{1}$$

where $\mathbf{A} = \mathbf{S}\mathbf{\Psi}^{-1}$ is a $k \times n$ *measurement* matrix that includes both the sampling and compression bases. If we could solve this linear system correctly for $\hat{\mathbf{x}}$ given $\mathbf{y}$, we could then recover the desired $\mathbf{x}$ by taking the inverse transform. Unfortunately, solving for $\hat{\mathbf{x}}$ 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 RIC condition is met (Sec. 3.2), then we can solve for $\hat{\mathbf{x}}$ uniquely by searching for the sparsest $\hat{\mathbf{x}}$ that solves the equation. A proof of this remarkable conclusion can be found in the paper by Candès et al. [38]. 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:

$$\min \|\hat{\mathbf{x}}\|_0 \text{ s.t. } \mathbf{y} = \mathbf{A}\hat{\mathbf{x}}. \tag{2}$$

Unfortunately, algorithms to solve Eq. 2 are NP-hard [41] because they involve a combinatorial search of all m-sparse vectors $\hat{\mathbf{x}}$ to find the sparsest one that meets the constraint. Fortunately, the CS research community has developed fast, greedy algorithms that find approximate solutions to the ℓ_0 problem, such as Orthogonal Matching Pursuit (OMP) [42]. Given the measured vector $\mathbf{y}$ and the measurement matrix $\mathbf{A}$, OMP finds the coefficient of $\hat{\mathbf{x}}$ 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 the largest coefficient has been

identified, we solve $\mathbf{y} = \mathbf{A}\hat{\mathbf{x}}$ as a least-squares problem assuming that it has only one non-zero coefficient in the identified position. We use this new estimate for $\hat{\mathbf{x}}$ to compute the estimated signal $\mathbf{x}$ and subtract it from the original measurements to get a residual. The algorithm iterates, using the residual to solve for the next largest coefficients of $\hat{\mathbf{x}}$ one at a time until an m-sparse approximation of the transform domain vector is found.

Despite its simplicity, OMP has a weak guarantee of exact recovery [43]. For this reason, Needell and Vershynin [43] 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 (see below). ROMP approximates the coefficients on every iteration like OMP and then sorts them in non-increasing order. Instead of selecting a single element in each iteration, ROMP selects the continuous set of coefficients with the largest energy in which the largest coefficient in the group cannot be more than twice as big as the smallest member. These coefficients are then appended to a list of non-zero coefficients maintained by the algorithm and a least-squares problem is solved to find the best approximation for these non-zero coefficients. The approximation error is then computed based on the measured results and the algorithm iterates again. Since we use the ROMP algorithm in this work, we provide a block diagram in Fig. 3 that gives an overview of the algorithm.

3.2 Restricted Isometry Condition (RIC)

It is impossible to solve $\mathbf{y} = \mathbf{A}\hat{\mathbf{x}}$ 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 $\hat{\mathbf{x}}$ if matrix $\mathbf{A}$ meets the Restricted Isometry Condition (RIC):

$$(1-\epsilon)||\mathbf{v}||_2 \leq ||\mathbf{A}\mathbf{v}||_2 \leq (1+\epsilon)||\mathbf{v}||_2, \tag{3}$$

with parameters (z, ϵ), where $\epsilon \in (0,1)$ for all z-sparse vectors $\mathbf{v}$ [43]. 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) [44].

Unfortunately, a point-sampled wavelet basis (which is desirable in our rendering application because of the ability for a rendering system to take point-samples of the scene) does not meet the RIC. We discuss our modifications to the basis to improve this and allow our framework to be used for rendering in the next section.

4 Compressive Rendering

In this section we present our framework by first describing the point-sampled wavelet measurement matrix for compressed sensing. We then describe a simple, non-adaptive algorithm based on the ROMP algorithm [43] introduced in Sec. 3.1 which solves for the missing pixels in the image.

4.1 Novel CS framework for point-sampling wavelets

Although wavelets are very good at compressing image data, they are unfortunately 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:

$$\mu(\mathbf{\Omega}, \mathbf{\Psi}) = \sqrt{n} \cdot \max_{1 \leq j,k \leq n} |\langle \omega_j, \psi_k \rangle|. \tag{4}$$

Because the matrices are orthonormal, the resulting coherence lies in the range $\mu(\mathbf{\Omega}, \mathbf{\Psi}) \in [1, \sqrt{n}]$ [45], 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 (see Fig. 7). 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. 1. Specifically, we assume that there exists a blurred image $\mathbf{x}_b$ which can be sharpened to form the original image: $\mathbf{x} = \mathbf{\Phi}^{-1}\mathbf{x}_b$, where $\mathbf{\Phi}^{-1}$ is a sharpening filter. We can now write the sampling process as $\mathbf{y} = \mathbf{S}\mathbf{x} = \mathbf{S}\mathbf{\Phi}^{-1}\mathbf{x}_b$. Since the blurred image $\mathbf{x}_b$ is also sparse in the wavelet domain, we can incorporate the wavelet compression basis in the same way as before and get $\mathbf{y} = \mathbf{S}\mathbf{\Phi}^{-1}\mathbf{\Psi}^{-1}\hat{\mathbf{x}}_b$. We can now solve for the sparsest $\hat{\mathbf{x}}_b$:

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

where $\mathbf{A} = \mathbf{S}\mathbf{\Phi}^{-1}\mathbf{\Psi}^{-1}$, using the greedy algorithms such as OMP or ROMP described in Sec. 3. Once $\hat{\mathbf{x}}_b$ has been found, we can compute our final image by taking the inverse wavelet transform and sharpening the result: $\mathbf{x} = \mathbf{\Phi}^{-1}\mathbf{\Psi}^{-1}\hat{\mathbf{x}}_b$.

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. 5, we get:

$$\min \|\hat{\mathbf{x}}_\mathbf{b}\|_0 \text{ s.t. } \mathbf{y} = \mathbf{S}\mathcal{F}^H\mathbf{G}^{-1}\mathcal{F}\mathbf{\Psi}^{-1}\hat{\mathbf{x}}_\mathbf{b}. \tag{6}$$

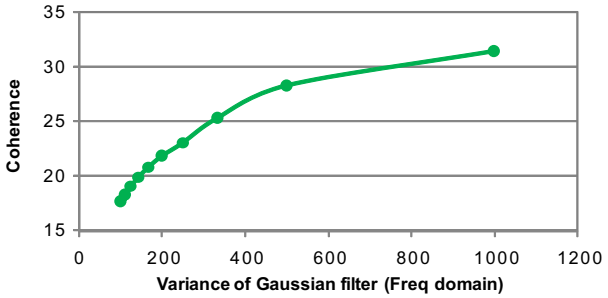


Fig. 2. Coherence vs. variance of Gaussian matrix $\mathbf{G}$. Since $\mathbf{G}$ is in the frequency domain, larger variance means a smaller spatial filter. As the variance grows, the coherence converges to 32, the coherence of the point-sampled, $64^2 \times 64^2$ DB-8 matrix. The coherence should 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 $\mathbf{x}_b$ and which is therefore harder to reconstruct accurately. The optimal values for the variance were determined empirically and decreased the coherence by almost a factor of 2.

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 [46], 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. 2 require a "backward" matrix $\mathbf{A}^\dagger$ that "undoes" the effect of $\mathbf{A}$ (i.e., $\|\mathbf{A}^\dagger\mathbf{A}\mathbf{v}\| \approx \|\mathbf{v}\|$), where $\mathbf{A} = \mathbf{S}\mathbf{\Phi}^{-1}\mathbf{\Psi}^{-1}$, we use a backwards matrix of the form $\mathbf{A}^\dagger = \mathbf{\Psi}\mathbf{\Phi}\mathbf{S}^T = \mathbf{\Psi}\mathcal{F}^H\mathbf{G}\mathcal{F}\mathbf{S}^T$.

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 $n\times k = 5.5\times 10^{11}$ elements. Therefore, our implementation must use a functional representation for $\mathbf{A}$ that can compute any element of $\mathbf{A}$ on the fly as needed. An explanation of how we do this can be found in Sec. 5.

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 $\mathbf{\Phi}^{-1}\mathbf{\Psi}^{-1}$ 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. 4 as a function of the variance σ^2 of Gaussian function of $\mathbf{G}$ in Fig. 2 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 $\mathbf{x}_b$ 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. We can gauge this effect by examining the results of *not* using the blur filter in Fig. 7. Furthermore, we can explore how the coherence vs. noise tradeoff affects image quality, results of which are shown in Fig. 9.

It is important to note that this framework still allows us to produce an image $\mathbf{x}$ which contains high frequency information which lies beyond the bandlimit of the filter $\mathbf{\Phi}$. As shown in Fig. 11 for a simple test case, high-frequencies beyond the apparent bandlimit of the filter can be recovered, although the reconstruction is biased towards the lower frequencies. With this new framework for filtered wavelet compression in place, we are now ready to apply it to our rendering application.

4.2 An algorithm for compressive rendering

The basic idea of our approach is simple: we use a ray tracer to fully compute a small number k of random pixel locations and then use compressed sensing to estimate the values of the other pixels assuming sparsity in the filtered wavelet basis introduced in the last section. This algorithm is based on the observation that the reconstructed images that deviate significantly from the fully-rendered image would also be less sparse in the transform domain and therefore are rejected by our algorithm. The non-adaptively sampled compressive rendering algorithm consists of two basic steps:

Step 1: The scene is rendered using standard ray tracing except that only k out of n pixels are computed. The k pixels to be rendered are randomly selected using a Poisson-disk distribution without taking into account scene properties. This step achieves the speedup over conventional rendering techniques, since not every pixel needs to be rendered.

Step 2: The measured pixels are represented as observation vector $\mathbf{y}$ in the compressed sensing framework of Sec. 3, which is then used to reconstruct the sparse wavelet transform $\hat{\mathbf{x}}_b$. This result is then used to compute the approximation of the desired image $\mathbf{x}$.

In step 1, we select a subset of the total pixels to be fully rendered. Note that our approach assumes that the pixels are independent from one another and can be computed separately. Although this independence is true of physical light transport, it is not necessarily true in rendering systems where samples from neighboring pixels might be used to compute the final pixel color. Therefore, we restrict the ray tracer to using samples only from inside the selected pixels, which is easy to do with stratified, jittered or low-discrepancy sampling. If done correctly, this first step is equivalent to rendering the entire image and then selecting the pixels at

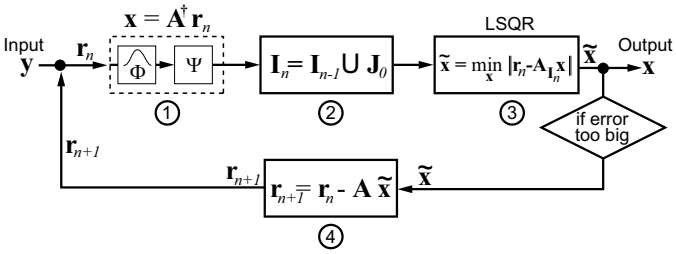


Fig. 3. Block diagram of the ROMP algorithm, which tries to solve $\mathbf{y} = \mathbf{A}\mathbf{x}$ with as few coefficients of $\mathbf{x}$ as possible. **1.** The input is first multiplied by $\mathbf{A}^{\dagger}$ to determine the largest coefficients of $\mathbf{x}$. In our formulation, this includes applying filter Φ and wavelet transform $\mathbf{\Psi}$. **2.** Next, the coefficients are sorted, and the set J_0 of contiguous coefficients with the largest energy is added to our index array I_n. **3.** We next solve the least-squares problem using only those coefficients to find the values for $\mathbf{x}$ that approximate the input when applied to the linear system. **4.** If the error is too big, the residual $\mathbf{r}_{n+1}$ is computed and the algorithm iterates again. Once the algorithm has converged or the target sparsity is reached, it outputs the approximation of $\mathbf{x}$.

the specified locations to form vector $\mathbf{y}$. Since the ray tracer only computes k pixels, we achieve a speedup over standard non-adaptive ray tracing of n/k . The k pixels to be computed are selected randomly, although we observe that random Poisson-disk distributions [47] work better than arbitrary random selection of the pixels because they do not clump together. Despite the fact the pixel samples are chosen without any information of the scene, we are still able to generate a high-quality rendering faster than conventional approaches.

Step 2 is the important reconstruction step, where the pixel samples from step 1 are reconstructed into the final image using the CS framework. We determine the color of the missing pixels by approximating a solution to Eq. 5 for $\hat{\mathbf{x}}_\mathbf{b}$ using Regularized Orthogonal Matching Pursuit (ROMP), which is shown as a block diagram in Fig. 3. Once $\hat{\mathbf{x}}_\mathbf{b}$ has been approximated, we can then estimate the desired final image $\mathbf{x}$ by taking the inverse wavelet transform of $\hat{\mathbf{x}}_\mathbf{b}$ and sharpening the result by applying the inverse of our Gaussian filter $\mathbf{\Phi}$. We now describe the details of our implementation in the next section.

5 Implementation and Results

To test our algorithm, we integrated our compressive sensing framework into both an academic ray tracing system (PBRT [5]) and a high-end, open source ray tracer (LuxRender [48]). The integration of both was straightforward since we only had to control the pixels being rendered (step 1), 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 [49]. 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

%	$1/\sigma^2$	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. See Sec. 4.1 for more details.

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. [43] available on their website but re-written in C++ for higher performance. We leverage the Intel Math Kernel Library 10.1 (MKL) libraries [50] to accelerate linear algebra computation and to perform the Fast-Fourier Transform for our Gaussian filter. In addition, we use the Stanford LSQR solver [51] to solve the least-squares step at the end of ROMP (see Fig. 3). 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 $\mathbf{A} = \mathbf{S}\mathcal{F}^H\mathbf{G}^{-1}\mathcal{F}\mathbf{\Psi}^{-1}$ from Eq. 6. The inverse wavelet transform $\mathbf{\Psi}^{-1}$ was computed using the lifting algorithm [14], 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 paper, 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 k, which has been observed to work well in the CS literature [52]. As shown in Fig. 3, ROMP uses the Gaussian filter 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 paper. Table 2 shows the actual values used in our experiments. If a sampling rate is used that

Scene	Pre-process	Full Render	CS Recon	%
ROBOTS	0.25	611	11.9	1.9%
WATCH	0.28	903	13.5	1.4%
GLASSES	0.46	470	13.3	2.8%
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, and photon mapping (for SPONZA which took 47 mins). **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.

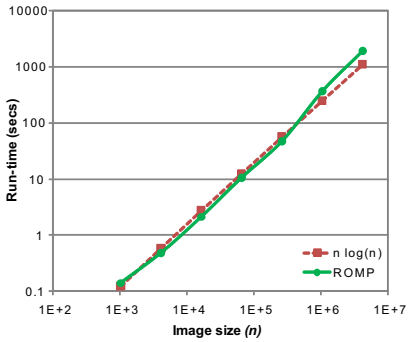


Fig. 4. Run-time complexity of our ROMP reconstruction algorithm, where n is the total size of our image (width $\times$ height). The curve of $n \log n$ is shown for comparison. We tested our algorithm on images of size 32×32 all the way through 2048×2048. Even at the larger sizes, the performance remained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

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 [53]. 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. [10], using the implementation by Alper and Mavinkurve [54]. To compare against approaches from the non-uniform sampling community, we implemented the Marvasti algorithm [37] 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 between by interpolating each triangle of the mesh, e.g., as described by Painter and Sloan [55]. 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. In terms of test scenes, we selected a variety of scenes that reflect the wide spectrum for high-end rendering: from scenes that might be used in feature films (ROBOTS), to architectural visualization – both indoor (APARTMENT) and outdoor (SPONZA), natural scenes (PLANTS), industrial rendering (WATCH), and finally artistic scenes (GLASSES). All final images shown in this paper are at 1000×1000 resolution except for the APARTMENT scene of Fig. 7, which is at 500×500. 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 margin.

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 implementation we

Scene	Interp	Cubic	ACT	Marvasti	Inpaint	CS
ROBOTS	2.00	4.94	1.98	2.11	4.24	1.72
WATCH	6.42	11	6.15	6.68	15	5.34
GLASSES	25	42	27	28	70	19
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.

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 apriori 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

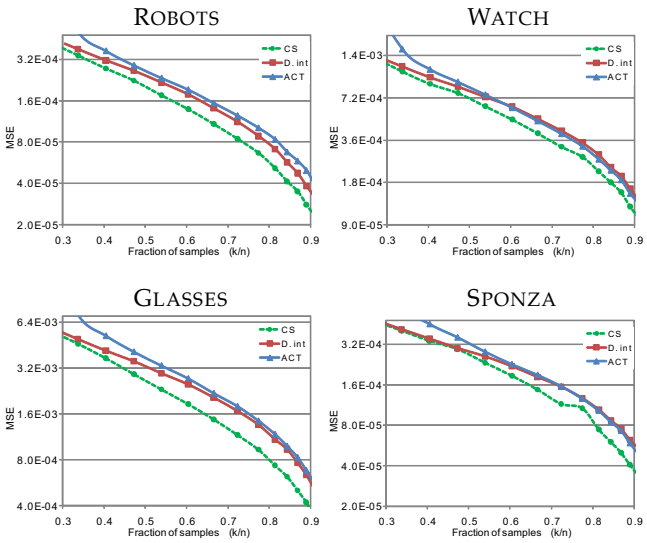


Fig. 5. Log error curves as a function of the number of samples for four test scenes using our technique and the best two other competing reconstruction algorithms. Our CS reconstruction beats both Delaunay-interpolation (D. int) as well as ACT, requiring 5% to 10% less samples to achieve a given level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

the Delaunay-interpolation used in rendering which works remarkably well.

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 four 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 Figs. 1, 6, and 7. 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 show some examples, we direct readers to the second inset of the ROBOTS scene in Fig. 6. 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 sam-

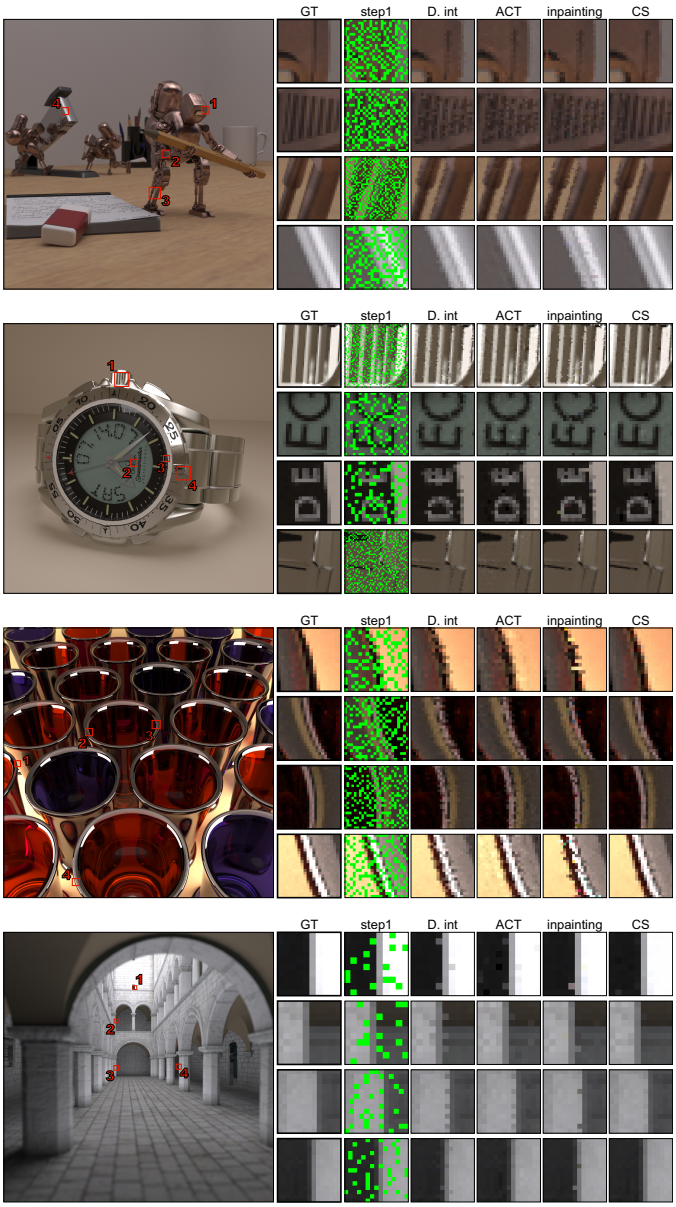


Fig. 6. Results of scenes with the reconstruction algorithms, each with a different % of computed samples. From top to bottom: ROBOTS (60%), WATCH (72%), GLASSES (67%), and SPONZA (87%). The large image shows the ground truth (GT) rendering to show context. The smaller columns show the ground truth (GT) of the inset region and the ray-traced pixels after step 1 (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.

ples 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.

6 Discussion

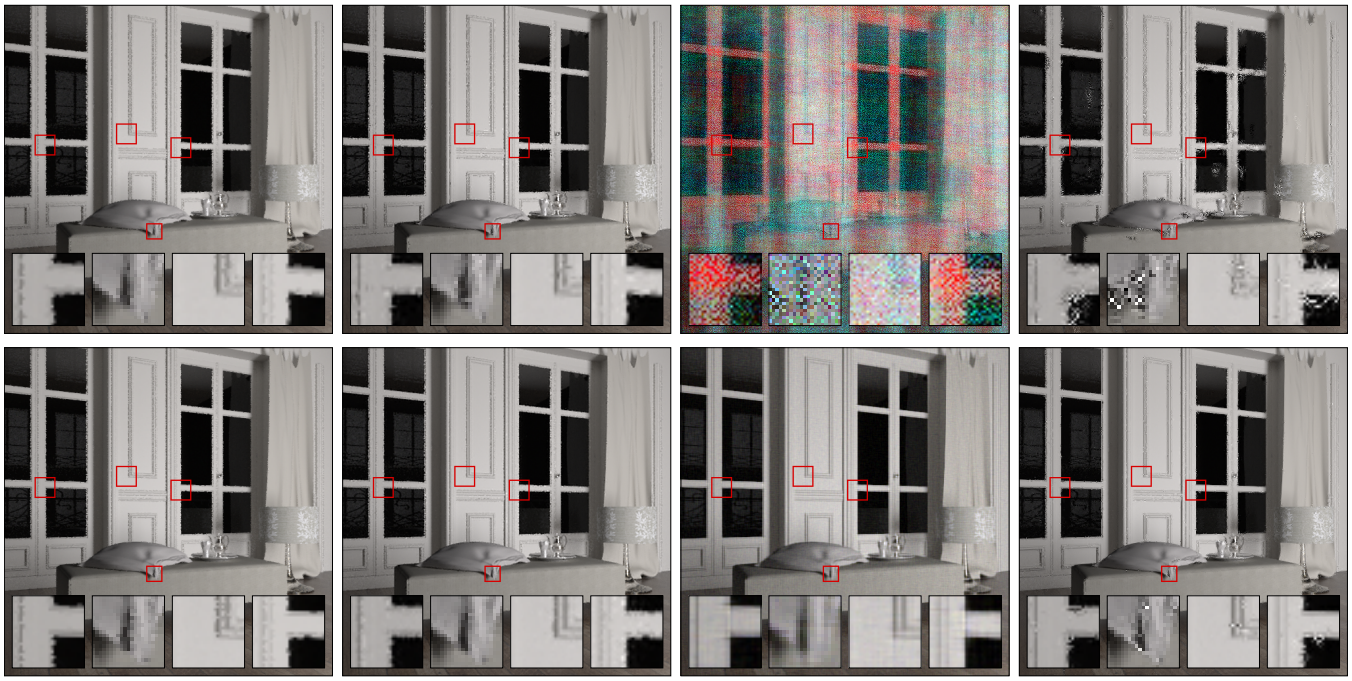
The proposed algorithm presents several avenues for interesting discussion, which we shall briefly touch on in this section.

6.1 Choice of compression basis

Since the results in this paper use the biorthogonal CDF 9/7 wavelet for compression, one might wonder how different basis would perform in our framework. It might seem at first glance that the Fourier basis would work because it is already maximally incoherent with point samples and therefore would not require our Gaussian filter (Sec. 4.1). When we tested this, however, we found that the poorer compression of the Fourier basis far outweighed the improved incoherency, as can be seen in Fig. 7. The image reconstructed from 45% of pixel samples assuming sparsity in the Fourier domain exhibits the strong ringing artifacts associated with artificially bandlimiting Fourier signals. We found that if we reduced the sampling rate even more, the image disintegrated altogether. In fact, we found that except for very simple images (i.e., images with large regions of constant color or slight color gradients), compressive rendering with a Fourier basis would perform worse than even the simple Delaunay interpolation. Therefore, we conclude that a Fourier basis for compression is not suitable for compressive rendering because images are simply not sparse enough in the Fourier domain in order for the algorithms to take advantage of this sparsity.

We can also examine the effect of using different wavelet basis for compression. To explore this, we tested our framework with two other wavelets (Daubechies-4, and Daubechies-8). Fig. 8 shows the error curves for one of our test scenes with these bases. We can see that the CDF 9/7 performs the best, but it is only a slight improvement over DB-8 and DB-4. In fact, there is relatively little visual difference between the results reconstructed using the different bases. The difference with interpolation, however, is considerably more pronounced. Our framework is unique in that it is the first to allow us to integrate the transform into the rendering process and compare the different compression bases in this way.

Furthermore, we can do experiments to check if the improved incoherency introduced by our blur filter Φ is making a difference by reconstructing the image using only a point-sampled wavelet basis without the Gaussian filter. As can be seen in the last image in Fig. 7, the resulting image has strong wavelet artifacts, something that we observed whenever we did not use the filter. This suggests that our proposed filter is helping to improve the reconstruction. We can think of the Gaussian filter as helping to bias the ROMP reconstruction towards lower frequency coefficients, therefore eliminating this high frequency ringing. This is similar to the observation by Tsaig and Donoho [52], where they bias the reconstruction towards specific coefficients.



Delaunay interpolation CS: Gaussian Wavelet CS: Fourier CS: Wavelet only

Fig. 7. To demonstrate the effectiveness of our Gaussian wavelet formulation, we test our technique (2nd column) on the APARTMENT scene (top row: 45% samples, bottom row: 60% samples) against a point-sampled Fourier basis which satisfies the RIC (3rd column) and a CDF 9/7 wavelet without the Gaussian blur (4th column). The Delaunay-interpolated reconstruction is shown for comparison (1st column). The image reconstructed using sparsity in the Fourier domain exhibits ringing typical of artificially bandlimiting the Fourier image and the quality is worse than that of the simple Delaunay interpolation. The reconstruction using the wavelet without the blur filter proposed in Sec. 4.1 also exhibits artifacts.

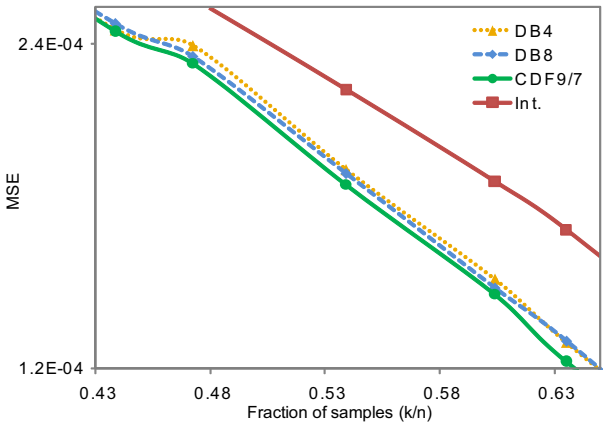


Fig. 8. Quality of the reconstruction when using different wavelet bases for compression for the ROBOTS scene. Of the transforms tested, CDF9/7 worked the best, although DB-8 and DB-4 were not far behind. These all had significant improvements over interpolation.

On a related note, we can measure the tradeoff between coherence and image noise by adjusting the variance of the filter $\mathbf{G}$. As mentioned in Sec. 4.1, the Gaussian filter $\mathbf{\Phi}$ that we add to our formulation helps reduce the coherence between the wavelet basis and the point samples to improve the quality of compressed sensing algorithms. However, filters with larger Gaussian kernels increase the noise in the final result because of the inversion process. Fig. 9 shows a series of images, starting a high-variance $\mathbf{G}$ on the left (which essentially means we use a spatial delta function for $\mathbf{\Phi}$) all the way to having a $\mathbf{G}$ with low variance (which blurs the image $\mathbf{x}_b$ considerably) on the right. The center image is at the empirical "sweet-spot" which we used for all our experiments.

6.2 Extension to adaptive sampling

As mentioned in Sec. 2.2, it is easy to use adaptive sampling with our algorithm by using the proposed CS reconstruction at every iteration to reconstruct the current version of the image. In other words, we take a small set of fixed samples, reconstruct the image using CS as described in Sec. 4.2, use the new image to determine the positions for the next set of samples, and repeat the process. Fig. 10 shows the result of doing this, using the contrast measure described by Mitchell [26] to decide where to place the new samples. Our approach continues to perform better than other reconstruction techniques.

6.3 Potential applications of our approach

Although our technique still requires a substantial fraction of the pixels to generate a good result, it could have useful applications in feature film production. For example, it could benefit pre-visualization, since having to render 30% less pixels when the average frame time is over 5 hours could result in considerable time savings. In

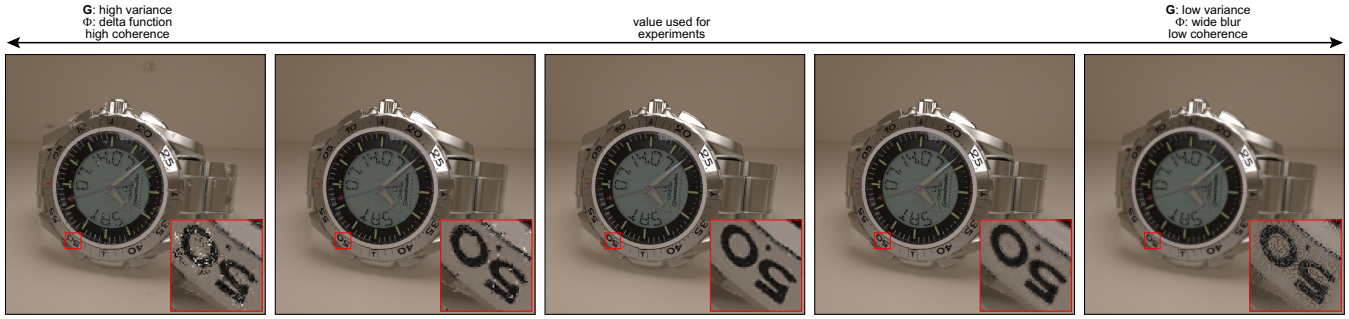


Fig. 9. The ideal Gaussian filter $\mathbf{G}$ should have a variance as small as possible in order to reduce coherence between the sampling and compression basis. If $\mathbf{G}$'s variance is too small, however, then the image $\mathbf{x}_b$ will be overly blurred (recall that $\mathbf{G}$ is in the frequency domain), which introduces noise into our final $\mathbf{x}$. In this figure, we show the quality of the reconstruction as we decrease the variance of $\mathbf{G}$ from left to right. On the leftmost end, $\mathbf{G}$ has a high variance which effectively means that we are using standard wavelets as our compression basis. We can see the artifacts due to coherence, which are similar to that of Fig. 7. As we move to the right, we reduce the variance gradually, moving through the sweet-spot (center image) until we decrease it enough to be overcome by the noise due to inversion of the filter (rightmost image). For this work, we experimentally determined the variance that yielded the best results on the ROBOTS scene, and then fixed that parameter for all data sets.

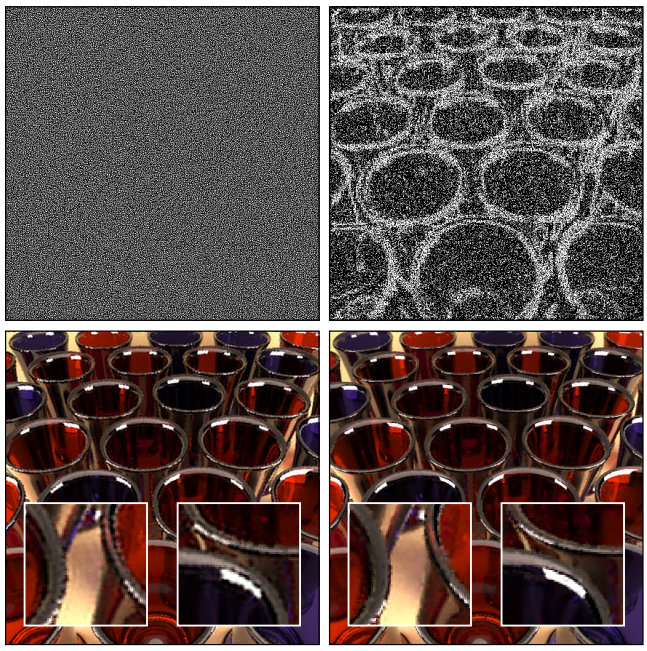


Fig. 10. Comparison of proposed non-adaptive (left) and its extension to an adaptive sampling algorithm (right). The top row shows the computed pixels (sampling rate: 33%). The non-adaptive algorithm (MSE: 8.66×10^{-3}) uses a random Poisson-disk distribution for placing the samples. As expected, the adaptive algorithm places more samples by the edge boundaries and performs better (MSE: 5.22×10^{-3}). If we use one of the other techniques for reconstruction (e.g., Delaunay-interpolation) the quality of the image is reduced (MSE: 5.40×10^{-3}).

pre-viz, time-savings are preferred over perfect quality, although better reconstruction algorithms are always preferable. Now that more CG films are being ray-traced (e.g., Sony Imageworks's "Cloudy with a Chance of Meatballs" [56]), the common practice is to render the dailies at lower resolution and simply upsample them for viewing by the director or lighting designers. Our algorithm can do better, as demonstrated by our results.

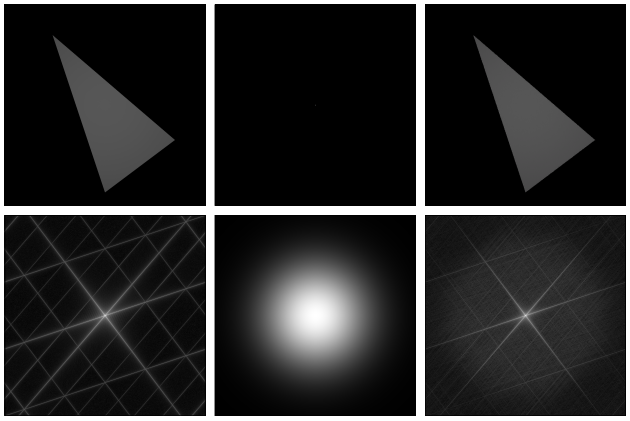
6.4 Limitations of the algorithm

In the process of studying compressive rendering, we have begun to understand some of its limitations. For example, it is not guaranteed to recover all the high-frequency content in an image. To see why, we can imagine that we have an image consisting of a single white pixel (a signal spike) in a constant black background. Clearly, we will be unable to recover this image correctly unless we measured this pixel directly with our initial samples. Furthermore, the addition of the Gaussian filter into our formulation (Sec. 4.1) biases the reconstruction towards lower frequencies, as can be seen in Fig. 11. This results in a difference between the original, ground-truth image and our approximate reconstruction. However, on the positive side, our results indicate that we can still recover certain high-frequency information, such as edges, more reliably than other techniques. This makes our approach still valuable for rendering.

Compressive rendering also fails at extremely low sampling densities ($< 5\%$ of pixel samples) where we currently cannot beat interpolation. At these low sampling rates, we do not have enough sparsity in the signal for ROMP to properly converge, since we need at least $5\times$ more samples than the support basis m. We also observe that the CDF $9/7$ wavelet is sensitive to the parameters of the Gaussian filter, more so than the Daubechies-8 (DB-8) wavelet we tested which also gives good results (see Fig. 8). If the Gaussian parameters for the CDF $9/7$ are set incorrectly, ROMP may not converge and the result looks like that in Fig. 12. It would be interesting to explore new basis functions that are not as sensitive to the filter and yet work well within our framework.

6.5 Future work

This work raises several interesting areas for future work. First of all, while we did not explicitly experiment with animated scenes, we have started to explore these



(a) (b) (c)

Fig. 11. Frequency content of original image and reconstruction. The spatial domain is shown in the top row, frequency domain in the bottom. (a) Original test image, which we can see contains high-frequency information. (b) Gaussian kernel Φ. (c) The resulting image after performing the reconstruction algorithm described in Sec. 4 using 45% of samples. Because of the way we pose the problem, we are able to recover frequencies beyond the apparent bandlimit of the filter, although we still see the impact of the filter in the bias towards lower frequencies in the reconstructed image.

ideas and it seems that our formulation can be extended to a 3-D wavelet which takes advantage of temporal coherence across a set of images. This is an advantage for our framework, since it is unclear how other methods (e.g., inpainting or ACT) could be extended in this way. It would also be interesting to explore the use of compressive rendering for depth-of-field and other effects, such as reflection and illumination models, since we can efficiently approximate the signal with less samples. In addition, the applied mathematics community is feverishly working on new CS-based reconstruction algorithms that work faster than ROMP and yield results of better quality. We look forward to their progress, since it will directly benefit the performance of our algorithm. Finally, it might also be possible to use this framework for other applications in computer graphics where we want to estimate missing information, such as inpainting.

7 Conclusion

In this paper, we introduce the idea of *compressive rendering*, in which sparsity in the transform-domain of an image is exploited to reduce rendering time by using it to estimate the values of the unrendered pixels. Our results indicate better performance than current algorithms that might be used to fill in the missing data. The proposed approach offers a new way of thinking about the sampling/reconstruction problem in computer graphics and might provide fruitful avenues for exploration in the future.

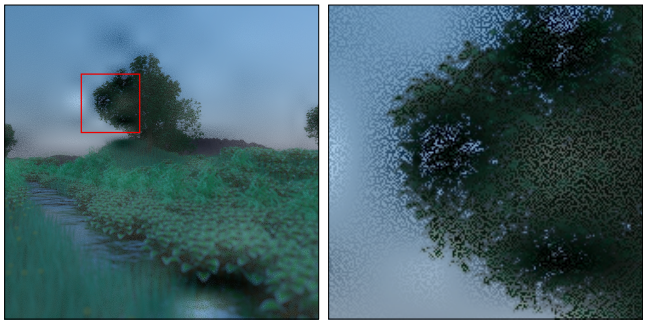


Fig. 12. Sensitivity of our algorithm to the Gaussian parameters. In this image of the scene PLANTS, the Gaussian parameters were set incorrectly so ROMP did not converge.

Acknowledgments

The authors would like to thank Dr. Yasamin Mostofi for valuable discussions during the course of this work. We also acknowledge the artists who provided us with the scenes used in the paper: ROBOTS by J. Peter Lloyd, WATCH and GLASSES by Luca Cugia, and the APARTMENT scene, which was modeled/textured by Bertrand Benoit with materials provided by Terrence Vargauwen. The SPONZA and PLANTS scenes are from the distribution of the PBRT raytracer by Pharr and Humphreys [5]. Finally, we thank Terrence Vargauwen for assistance with integrating our framework with LuxRender, which is available open-source from their website [48]. This work was sponsored by the NSF CAREER Award # 0845396 "A Framework for Sparse Signal Reconstruction for Computer Graphics."

References

[1] T. Whitted, "An improved illumination model for shaded display," *Communications of the ACM*, vol. 23, no. 6, pp. 343–349, 1980.

[2] R. L. Cook, L. Carpenter, and E. Catmull, "The REYES image rendering architecture," *SIGGRAPH Comput. Graph.*, vol. 21, no. 4, pp. 95–102, 1987.

[3] M. Segal and K. Akeley, "The OpenGL Graphics System: A specification, v. 1.0," Silicon Graphics, Tech. Rep., 1992.

[4] M. Bolin and G. Meyer, "A frequency based ray tracer," in *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, 1995, pp. 409–418.

[5] M. Pharr and G. Humphreys, *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., 2004.

[6] J. Gu, S. Nayar, E. Grinspun, P. Belhumeur, and R. Ramamoorthi, "Compressive structured light for recovering inhomogeneous participating media," in *ECCV*, Oct 2008.

[7] P. Peers, D. Mahajan, B. Lamond, A. Ghosh, W. Matusik, R. Ramamoorthi, and P. Debevec, "Compressive light transport sensing," *ACM Trans. Graph.*, vol. 28, no. 1, pp. 1–18, 2009.

[8] P. Sen and S. Darabi, "Compressive Dual Photography," *Computer Graphics Forum*, vol. 28, no. 2, pp. 609 – 618, 2009.

[9] H. Feichtinger, K. Gröchenig, and T. Strohmer, "Efficient numerical methods in non-uniform sampling theory," *Numer. Math*, vol. 69, pp. 423–440, 1995.

[10] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester, "Image inpainting," in *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 2000, pp. 417–424.

[11] P. Schröder, "Wavelets in computer graphics," *Proceedings of the IEEE*, vol. 84, no. 4, pp. 615–625, Apr 1996.

[12] P. Schröder and W. Sweldens, "Wavelets in computer graphics," *SIGGRAPH 1996 Course Notes*, no. 13, 1996.

[13] E. J. Stollnitz, T. D. Derose, and D. H. Salesin, *Wavelets for computer graphics: theory and applications*. Morgan Kaufmann Publishers Inc., 1996.
[14] S. Mallat, *A Wavelet Tour of Signal Processing*, 2nd ed. Academic Press, 1999.
[15] P. Hanrahan, D. Salzman, and L. Aupperle, "A rapid hierarchical radiosity algorithm," *SIGGRAPH Comput. Graph.*, vol. 25, no. 4, pp. 197–206, 1991.
[16] S. J. Gortler, P. Schröder, M. F. Cohen, and P. Hanrahan, "Wavelet radiosity," in *SIGGRAPH '93*, 1993, pp. 221–230.
[17] D. Lischinski, F. Tampieri, and D. P. Greenberg, "Combining hierarchical radiosity and discontinuity meshing," in *SIGGRAPH*, 1993, pp. 199–208.
[18] P. Schröder, S. J. Gortler, M. F. Cohen, and P. Hanrahan, "Wavelet projections for radiosity," *Computer Graphics Forum*, vol. 13, no. 2, 1994.
[19] R. Ramamoorthi and P. Hanrahan, "An efficient representation for irradiance environment maps," in *SIGGRAPH*, 2001.
[20] P.-P. Sloan, J. Kautz, and J. Snyder, "Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments," in *SIGGRAPH*, 2002, pp. 527–536.
[21] R. Ng, R. Ramamoorthi, and P. Hanrahan, "All-frequency shadows using non-linear wavelet lighting approximation," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 376–381, 2003.
[22] ——, "Triple product wavelet integrals for all-frequency relighting," in *SIGGRAPH*, 2004, pp. 477–487.
[23] T. Malzbender, "Fourier volume rendering," *ACM Trans. Graph.*, vol. 12, no. 3, pp. 233–250, 1993.
[24] T. Totsuka and M. Levoy, "Frequency domain volume rendering," in *SIGGRAPH*, 1993, pp. 271–278.
[25] M. H. Gross, L. Lippert, R. Dittrich, and S. Hring, "Two methods for wavelet-based volume rendering," *Computers and Graphics*, vol. 21, no. 2, pp. 237 – 252, 1997.
[26] D. P. Mitchell, "Generating antialiased images at low sampling densities," in *SIGGRAPH*, 1987, pp. 65–72.
[27] B. Walter, G. Drettakis, and S. Parker, "Interactive rendering using the render cache," in *Proceedings of the 10th Eurographics Workshop on Rendering*, D. Lischinski and G. Larson, Eds., vol. 10. New York, NY: Springer-Verlag/Wien, Jun 1999, pp. 235–246.
[28] P. Tole, F. Pellacini, B. Walter, and D. Greenberg, "Interactive global illumination in dynamic scenes," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 537–546, 2002.
[29] F. Pighin, D. Lischinski, and D. Salesin, "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, pp. 115–125.
[30] B. Guo, "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, pp. 255–266.
[31] K. Bala, B. Walter, and D. P. Greenberg, "Combining edges and points for interactive high-quality rendering," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 631–640, 2003.
[32] P. Sen, M. Cammarano, and P. Hanrahan, "Shadow silhouette maps," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 521–526, 2003.
[33] P. Sen, "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, pp. 65–73.
[34] T. Hachisuka, W. Jarosz, R. P. Weistroffer, K. Dale, G. Humphreys, M. Zwicker, and H. W. Jensen, "Multidimensional adaptive sampling and reconstruction for ray tracing," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 1–10, 2008.
[35] S. Masnou and J.-M. Morel, "Level lines based disocclusion," in *Proceedings of ICIP*, Oct 1998, pp. 259–263.
[36] F. Marvasti, *Nonuniform Sampling: Theory and Practice*. Kluwer Academic Publishers, 2001.
[37] F. Marvasti, C. Liu, and G. Adams, "Analysis and recovery of multidimensional signals from irregular samples using nonlinear and iterative techniques," *Signal Process.*, vol. 36, no. 1, pp. 13–30, 1994.
[38] E. J. Candès, J. Romberg, and T. Tao, "Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information," *IEEE Trans. on Information Theory*, vol. 52, no. 2, pp. 489–509, Feb. 2006.
[39] D. L. Donoho, "Compressed sensing," *IEEE Trans. on Information Theory*, vol. 52, no. 4, pp. 1289–1306, Apr. 2006.
[40] "Rice University Compressive Sensing Resources website," 2009, http://www.dsp.ece.rice.edu/cs/.
[41] E. J. Candès, M. Rudelson, T. Tao, and R. Vershynin, "Error correction via linear programming." in *IEEE Symposium on Foundations of Computer Science*, 2005, pp. 295–308.
[42] J. A. Tropp and A. C. Gilbert, "Signal recovery from random measurements via orthogonal matching pursuit," *IEEE Trans. on Information Theory*, vol. 53, no. 12, pp. 4655–4666, Dec. 2007.
[43] D. Needell and R. Vershynin, "Uniform uncertainty principle and signal recovery via regularized orthogonal matching pursuit," 2007, preprint.
[44] E. J. Candès and T. Tao, "Near optimal signal recovery from random projections: universal encoding strategies?" *IEEE Trans. on Information Theory*, vol. 52, no. 12, pp. 5406–5425, Dec. 2006.
[45] D. L. Donoho and X. Huo, "Uncertainty principles and ideal atomic decomposition," *IEEE Transactions on Information Theory*, vol. 47, no. 7, pp. 2845–2862, Nov 2001.
[46] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Addison-Wesley Longman Publishing Co., Inc., 2001.
[47] R. L. Cook, "Stochastic sampling in computer graphics," *ACM Trans. Graph.*, vol. 5, no. 1, pp. 51–72, 1986.
[48] "LuxRender," http://www.luxrender.net/.
[49] D. Dunbar and G. Humphreys, "A spatial data structure for fast Poisson-disk sample generation," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 503–508, 2006.
[50] "Intel Math Kernel Library," 2009, http://www.intel.com/.
[51] "Stanford Systems Optimization Laboratory," 2009, http://www.stanford.edu/group/SOL/software/lsqr.html.
[52] Y. Tsaig and D. L. Donoho, "Extensions of compressed sensing," *Signal Process.*, vol. 86, no. 3, pp. 549–571, 2006.
[53] D. S. Taubman and M. W. Marcellin, *JPEG 2000: Image Compression Fundamentals, Standards and Practice*. Springer, 2001.
[54] E. Alper and S. Mavinkurve, "Image inpainting," 2002, http://www.eecs.harvard.edu/ sanjay/inpainting/.
[55] J. Painter and K. Sloan, "Antialiased ray tracing by adaptive progressive refinement," *SIGGRAPH Comput. Graph.*, vol. 23, no. 3, pp. 281–288, 1989.
[56] *Cloudy with a Chance of Meatballs*. Sony Pictures Animation, 2009, http://www.cloudywithachanceofmeatballs.com/.

Pradeep Sen is an Assistant Professor in the Department of Electrical and Computer Engineering at the University of New Mexico. He received his B.S. in Computer and Electrical Engineering from Purdue University in 1996 and his M.S. in Electrical Engineering from Stanford University in 1998 in the area of electron-beam lithography. After two years of working at a profitable startup company which he co-founded, he joined the Stanford Graphics Lab in the Fall of 2000 where he received his Ph.D. in Electrical Engineering in June 2006. His interests in computer graphics include real-time graphics and graphics hardware, global illumination algorithms, computational photography and display technology. Dr. Sen has been with the University of New Mexico since the Fall of 2006, where he founded the UNM Advanced Graphics Lab, a laboratory dedicated to research in computer graphics and vision. Dr. Sen was awarded an NSF CAREER award in 2009 to study applications of compressed sensing to computer graphics.

Soheil Darabi is a Ph.D. candidate in the Department of Electrical and Computer Engineering at the University of New Mexico. He received his B.S. in 2005 and his M.S. in 2007 in Computer Engineering from Sharif University in Iran. He joined the UNM Advanced Graphics Lab in the Fall of 2007. Soheil's research interests include the application of signal processing theory to photorealistic and real-time rendering systems.