



ARL-TR-8876 • DEC 2019



# An Introduction to Computational Ghost Imaging with Example Code

by Michael Don

Approved for public release; distribution is unlimited.

## **NOTICES**

### **Disclaimers**

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.



# **An Introduction to Computational Ghost Imaging with Example Code**

**Michael Don**

*Weapons and Materials Research Directorate, CCDC Army Research Laboratory*

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
<p>Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.</p> <p><b>PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.</b></p>					
1. REPORT DATE (DD-MM-YYYY)		2. REPORT TYPE		3. DATES COVERED (From - To)	
December 2019		Technical Report		December 2018–January 2019	
4. TITLE AND SUBTITLE An Introduction to Computational Ghost Imaging with Example Code				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Michael Don				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) CCDC Army Research Laboratory ATTN: FCDD-RLW-LF Aberdeen Proving Ground, MD 21005				8. PERFORMING ORGANIZATION REPORT NUMBER  ARL-TR-8876	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.					
13. SUPPLEMENTARY NOTES ORCID ID: Michael Don, 0000-0002-8021-9066					
14. ABSTRACT Ghost imaging (GI) is a novel technique where an object is indirectly imaged using a bucket detector to collect light reflected from or transmitted through an object. This is particularly useful in harsh environments where traditional imaging will fail. This report introduces and compares a number of GI techniques, including traditional GI, computational GI and its variants, structured illumination, and denoising. In addition, compressive sensing reconstruction is presented and compared with GI. Example MATLAB code is provided to reproduce many of the report's simulations.					
15. SUBJECT TERMS ghost imaging, compressive sensing, single-pixel imaging, computational imaging, denoising					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT  UU	18. NUMBER OF PAGES  52	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Michael Don
Unclassified	Unclassified	Unclassified			19b. TELEPHONE NUMBER (Include area code) 410-306-0775

## **Contents**

---

<b>List of Figures</b>	<b>iv</b>
<b>1. Introduction</b>	<b>1</b>
<b>2. Traditional Ghost Imaging</b>	<b>1</b>
<b>3. Computational Ghost Imaging</b>	<b>4</b>
<b>4. Structured Illumination</b>	<b>12</b>
<b>5. Denoising</b>	<b>21</b>
<b>6. Compressive Sensing</b>	<b>22</b>
<b>7. Conclusion</b>	<b>30</b>
<b>8. References</b>	<b>31</b>
<b>Appendix. Example MATLAB Code</b>	<b>34</b>
<b>List of Symbols, Abbreviations, and Acronyms</b>	<b>44</b>
<b>Distribution List</b>	<b>45</b>

## List of Figures

Fig. 1	Traditional ghost imaging .....	2
Fig. 2	$32 \times 32$ test image .....	3
Fig. 3	Example recovered images using GI over a range of sampling ratios..	4
Fig. 4	GI PSNR vs. sampling ratio.....	4
Fig. 5	CGI setup .....	4
Fig. 6	Comparison of CGI and TGI for various noise ratios $\kappa$ : top row is CGI and bottom row is TGI.....	6
Fig. 7	Comparison of CGI and TGI over a range of noise ratios $\kappa$ .....	6
Fig. 8	Diagram of DGI .....	7
Fig. 9	Execution time of GI reconstruction algorithms vs. sampling ratio .....	8
Fig. 10	Comparison of CGI, DGI, NGI, and PINV for several sampling ratios	9
Fig. 11	Comparison of CGI, DGI, NGI, and PINV with scaled illumination for several sampling ratios.....	10
Fig. 12	Summary of the nonscaled and scaled simulation results for the simulations in Figs. 10 and 11 .....	11
Fig. 13	Performance comparison using scaled illumination including the new DPINV and NPINV algorithms .....	12
Fig. 14	Example DCT illumination patterns .....	13
Fig. 15	Example image, the DCT of the image, and a high-contrast visualization of the DCT. The lower frequencies of the DCT occur in the upper left and generally have the largest amplitudes. ....	13
Fig. 16	Comparison of the image pixel magnitudes in the standard basis, the DCT coefficient magnitudes, and the WHT coefficient magnitudes..	14
Fig. 17	Zig-zag method used to order DCT coefficients for JPEG compression .....	14
Fig. 18	Comparison of CGI, DGI, and PINV using DCT illumination for several sampling ratios.....	15
Fig. 19	Example Hadamard illumination patterns .....	16
Fig. 20	Comparison of CGI, DGI, NGI, and PINV using Hadamard illumination for several sampling ratios .....	17
Fig. 21	Example noiselet illumination patterns.....	18
Fig. 22	Comparison of CGI, DGI, NGI, and PINV using noiselet illumination for several sampling ratios .....	19
Fig. 23	A performance summary comparing DCT, noiselet, Hadamard, and binary illumination using CGI, DGI, NGI, and PINV image recovery .....	20

Fig. 24	Performance summary comparing CGI, DGI, NGI, and PINV image recovery algorithms for DCT, noiselet, Hadamard, and binary illumination .....	20
Fig. 25	Performance gain of denoising an image recovered through CGI over a range of sampling ratios.....	22
Fig. 26	Examples of denoising CGI images for 3 sampling ratios .....	22
Fig. 27	Comparison of CGI with noiselet and Hadamard illumination, and CS using a DCT basis and TV minimization.....	24
Fig. 28	Example images comparing CGI with noiselet and Hadamard illumination, and CS using a DCT basis and TV minimization .....	24
Fig. 29	Execution time of the CS algorithm using the DCT basis .....	25
Fig. 30	Comparison of GI and CS using the image sparse in the standard basis shown in Fig. 31.....	26
Fig. 31	Example images comparing GI with CS using a sparse image in the standard basis .....	27
Fig. 32	Comparison between GI and CS using the image sparse in the DCT basis shown in Fig. 33.....	28
Fig. 33	Example high-contrast images comparing GI with CS using a sparse image in the DCT basis.....	29

## 1. Introduction

---

Ghost imaging (GI) is a novel imaging technique where an object is imaged using two light sources: an object beam that illuminates the object and is measured by a bucket detector, and a reference beam measured by a high-resolution focal point array (FPA).<sup>1</sup> The light collected by the bucket detector interacts with the object but has no resolution. The reference beam is detected by a high-resolution imager, but does not interact with the object. Although neither measurements can image the object alone, a ghost image of the object can be created by combining the measurements. GI is applicable to imaging objects in harsh environments where typical imaging techniques will fail.

Initial work on GI was posited to use quantum entanglement between the photons of the object and reference beams.<sup>2,3</sup> Later work demonstrated GI without quantum entanglement and without even a reference beam.<sup>4,5</sup> This technique is called computational GI (CGI). It is beyond the scope of this report to explore the role of quantum entanglement in GI, hence only classical optical effects are considered.

Several papers offer a review of GI,<sup>6-9</sup> but many of them focus on the issue of quantum entanglement, and are either too detailed for a basic introduction or too simple, focusing on the history of GI without addressing GI algorithms. In addition, the author is not aware of any GI papers accompanied by open-source software to allow for reproducible results. This report seeks to fill this gap, presenting and comparing several GI techniques and providing example MATLAB code. Traditional GI (TGI) is explained, modeled, and simulated. Basic CGI is introduced and compared with TGI. Building on basic CGI, several CGI variants are presented, and their performance is compared. GI with structured and pseudo-random illumination is evaluated. The enhancement of GI by denoising is demonstrated. Finally, compressive sensing (CS) image reconstruction is presented and compared to GI reconstruction.

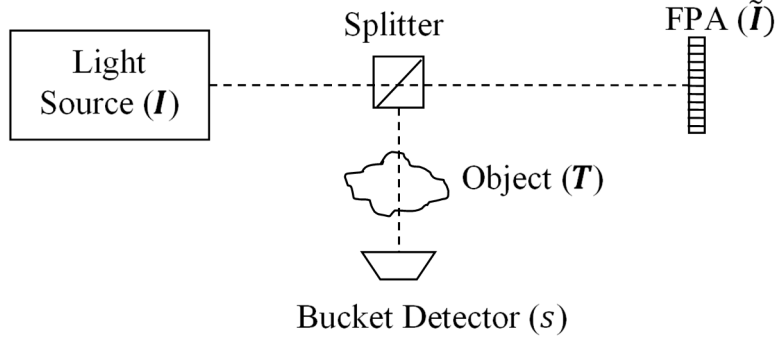
## 2. Traditional Ghost Imaging

---

Figure 1 shows a diagram of TGI.<sup>1</sup> A light source transmits an intensity pattern modeled as matrix  $\mathbf{I} \in \mathbb{R}^{n \times n}$ . The light is split, part detected by a  $n \times n$  FPA as values  $\tilde{\mathbf{I}} \in \mathbb{R}^{n \times n}$  and part transmitted through a partially transparent object modeled as transparency matrix  $\mathbf{T} \in \mathbb{R}^{n \times n}$ . GI can also be performed using light reflected from opaque objects. A single-pixel bucket detector sums the light transmitted through the object, producing a scalar measurement  $s$ . This basic diagram ignores lenses that some implementations may require. It is assumed that there is some environmental factor that prevents direct imaging with the FPA but allows the



bucket detector to collect the object's light. The measurement process is repeated to produce  $M$  measurements, each using different illumination patterns. The light might be a random, uncontrolled source, such as a pseudothermal source, or it may be a controlled spatial light modulator (SLM).



**Fig. 1 Traditional ghost imaging**

An image of the object,  $\mathbf{O} \in \mathbb{R}^{n \times n}$ , is recovered by calculating the correlation between  $\tilde{I}$  and  $s$  as

$$\mathbf{O} = \langle (\tilde{I} - \langle \tilde{I} \rangle)(s - \langle s \rangle) \rangle, \quad (1)$$

where  $\langle \cdot \rangle$  denotes an average value. More explicitly, using  $i$  as the measurement index,

$$\mathbf{O} = \frac{1}{M} \sum_{i=1}^M \left( (\tilde{I}_i - \langle \tilde{I} \rangle)(s_i - \langle s \rangle) \right), \quad (2)$$

with

$$\langle \tilde{I} \rangle = \frac{1}{n} \sum_{i=1}^n \tilde{I}_i \quad (3)$$

and

$$\langle s \rangle = \frac{1}{n} \sum_{i=1}^n s_i. \quad (4)$$

In a compact notation, the measurement process can be modeled as

$$\mathbf{s} = \mathbf{A}\mathbf{x}, \quad (5)$$

where  $\mathbf{A} \in \mathbb{R}^{M \times N}$  is the measurement matrix with rows

$$\mathbf{a}_i = \text{vec}(\tilde{I}_i)^T, \quad (6)$$

$\mathbf{x} \in \mathbb{R}^{N \times 1}$  is the vecorized object

$$\mathbf{x} = \text{vec}(\mathbf{T}), \quad (7)$$

and  $\mathbf{s} \in \mathbb{R}^{M \times 1}$  is the measurement vector

$$\mathbf{s} = (s_1, s_2, \dots, s_M)^T, \quad (8)$$

where  $N = n^2$  and  $\text{vec}(\cdot)$  is an operator that vectorizes a matrix into a single column. This simple model excludes any path loss, optical effects, or noise; therefore,  $\tilde{\mathbf{I}}$  represents the true modeled object illumination. Using this measurement matrix notation, Eq. 1 can be rewritten as

$$\tilde{\mathbf{x}} = (\mathbf{A} - \text{vec}(\langle \tilde{\mathbf{I}} \rangle) \mathbf{1}^T)^T (\mathbf{y} - \langle \mathbf{s} \rangle), \quad (9)$$

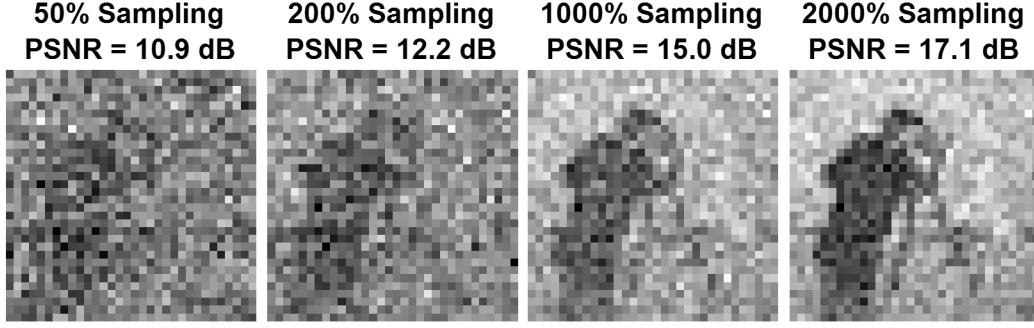
with  $\tilde{\mathbf{x}} = \text{vec}(\mathbf{O})$ . The  $\text{vec}(\langle \tilde{\mathbf{I}} \rangle) \mathbf{1}^T$  term performs mean centering, subtracting  $\text{vec}(\langle \tilde{\mathbf{I}} \rangle)^T$  from every row of  $\mathbf{A}$ , where  $\mathbf{1} \in \mathbb{R}^{N \times 1}$  is a vectors of ones.

Using the  $32 \times 32$  test image in Fig. 2, GI measurements were simulated using Eq. 5, with a random binary  $\mathbf{I}$  (i.e., the elements of  $\mathbf{I}$  were drawn from a Bernoulli random variable with equal probabilities of 0 and 1). The simulation was noiseless, with  $\tilde{\mathbf{I}} = \mathbf{I}$  and no noise added to the measurement vector  $\mathbf{s}$ . Figures 3 and 4 show the results of TGI recovery using Eq. 9 for a range of sampling ratios. The sampling ratio in percent is defined as  $100M/N$ . Both the original and recovered images were normalized so their values were in the interval  $[0,1]$  before calculating the peak signal to noise ratio (PSNR). The rightmost image in Fig. 3 shows the recovered image using a sampling ratio of 2,000%. Although recognizable, the image quality of TGI is still poor after using a large amount of measurements.

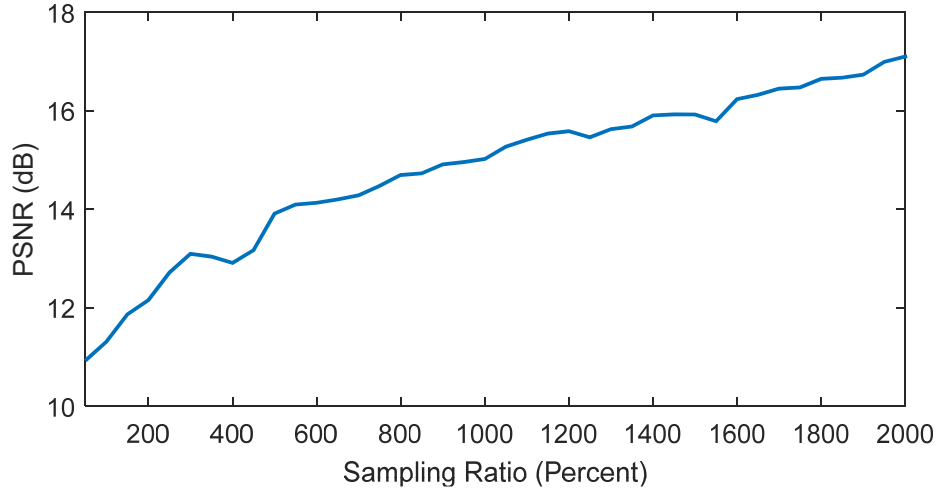
**Original Image**



**Fig. 2**  $32 \times 32$  test image



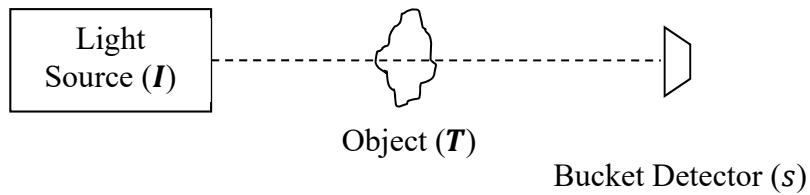
**Fig. 3** Example recovered images using GI over a range of sampling ratios



**Fig. 4** GI PSNR vs. sampling ratio

### 3. Computational Ghost Imaging

In TGI, the measured value of  $\tilde{I}$  is used to recover an image of the object. In CGI, instead of measuring  $\tilde{I}$ , it is assumed that  $\tilde{I} = I$ .<sup>4</sup> This is only possible when using an SLM where  $I$  can be predetermined. This allows us to dispense with the FPA, leading to the simplified CGI experiment setup in Fig. 5. In the previous TGI simulation it was also assumed that  $\tilde{I} = I$ ; therefore, a CGI simulation without noise will have the same results as those shown in Figs. 3 and 4. To adequately compare TGI and CGI, a more-sophisticated measurement model is needed.



**Fig. 5** CGI setup

In the discussion on TGI, the source illumination  $\mathbf{I}$  has already been distinguished from the measured illumination  $\tilde{\mathbf{I}}$ . The desired ideal illumination pattern  $\mathbf{I}$  is now distinguished from the actual illumination  $\mathbf{I}'$ .  $\mathbf{I}'$  is modeled with a linear scale factor and Gaussian noise  $\mathbf{v}_{I'} \in \mathbb{R}^{n \times n}$  as

$$\mathbf{I}' = (bi + c)\mathbf{I} + \mathbf{v}_{I'} \quad (10)$$

and used in place of  $\tilde{\mathbf{I}}$  in the construction of the measurement matrix in Eq. 6. Optical effects such as the point spread function and attenuation are ignored.  $\tilde{\mathbf{I}}$  is derived from  $\mathbf{I}'$  with additive noise  $\mathbf{v}_{\tilde{\mathbf{I}}} \in \mathbb{R}^{n \times n}$

$$\tilde{\mathbf{I}} = \mathbf{I}' + \mathbf{v}_{\tilde{\mathbf{I}}}. \quad (11)$$

Measurement noise  $\mathbf{v}_s \in \mathbb{R}^{M \times 1}$  is added to Eq. 5 giving

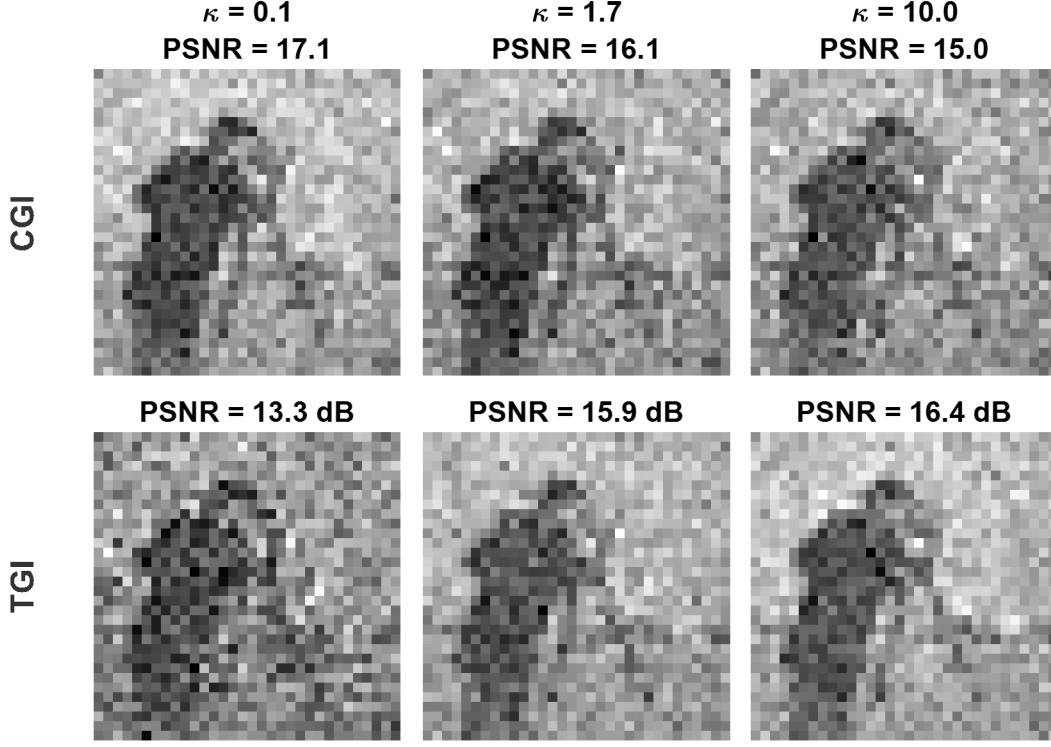
$$\mathbf{s} = \mathbf{A}\mathbf{x} + \mathbf{v}_s. \quad (12)$$

Since TGI directly measures  $\tilde{\mathbf{I}}$ , the scaling and noise of  $\mathbf{I}'$  can be accounted for in GI image reconstruction. In CGI,  $\tilde{\mathbf{I}}$  is not measured. Therefore, the bias and noise of  $\mathbf{I}'$  become extra system noise, degrading reconstruction performance.

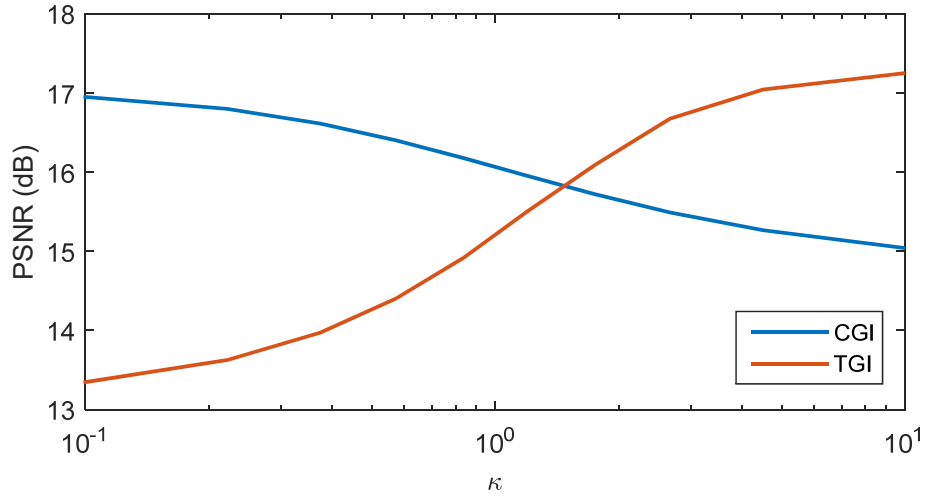
Figures 6 and 7 show a comparison of TGI and CGI for various noise levels using a fixed sampling ratio of 2,000% and a random binary illumination pattern. The noise levels were determined using a noise ratio:

$$\kappa = \frac{\sigma_s + \sigma_{I'}}{\sigma_{\tilde{\mathbf{I}}}}, \quad (13)$$

where  $\sigma_s$ ,  $\sigma_{I'}$ , and  $\sigma_{\tilde{\mathbf{I}}}$  are the standard deviations of  $\mathbf{v}_s$ ,  $\mathbf{v}_{I'}$ , and  $\mathbf{v}_{\tilde{\mathbf{I}}}$  respectively. No scale factor was used for this simulation (i.e.,  $b = c = 0$ ). The values of  $\sigma_s$  and  $\sigma_{I'}$  were kept equal, as was the total value of  $\sigma_s + \sigma_{I'} + \sigma_{\tilde{\mathbf{I}}} = 1.1$ . When  $\kappa$  is small, the measured illumination noise  $\mathbf{v}_{\tilde{\mathbf{I}}}$  dominates, creating a substantial difference between the actual object illumination  $\mathbf{I}'$  and the measured illumination  $\tilde{\mathbf{I}}$ . Since TGI relies on  $\tilde{\mathbf{I}}$  for image reconstruction, we expect that in this case TGI would perform poorly. On the other hand, CGI relies on the ideal illumination  $\mathbf{I}$  for reconstruction. Since  $\mathbf{v}_{I'}$  is small,  $\mathbf{I}$  is close to  $\mathbf{I}'$ , leading to high-quality CGI images. When  $\kappa$  is large,  $\mathbf{v}_{\tilde{\mathbf{I}}}$  is small and  $\mathbf{v}_{I'}$  is large, leading to the opposite result where TGI outperforms CGI. In many practical applications, we expect that  $\mathbf{v}_{\tilde{\mathbf{I}}}$  is relatively small and that TGI will perform better than CGI. The advantage of CGI is that it operates without the FPA, simplifying the measurement process.



**Fig. 6** Comparison of CGI and TGI for various noise ratios  $\kappa$ : top row is CGI and bottom row is TGI



**Fig. 7** Comparison of CGI and TGI over a range of noise ratios  $\kappa$

There are a number of variations of GI image recovery. Some are applicable to traditional GI as well as CGI, but the remainder of this report will focus exclusively on CGI. Figure 8 shows a variant of CGI called differential GI (DGI).<sup>10</sup> Like CGI, the ideal illumination  $\mathbf{I}$  is used for reconstruction, but a reference bucket detector

is employed to measure the total power of the light source. Using the form of Eq. 1, DGI reconstruction is given by

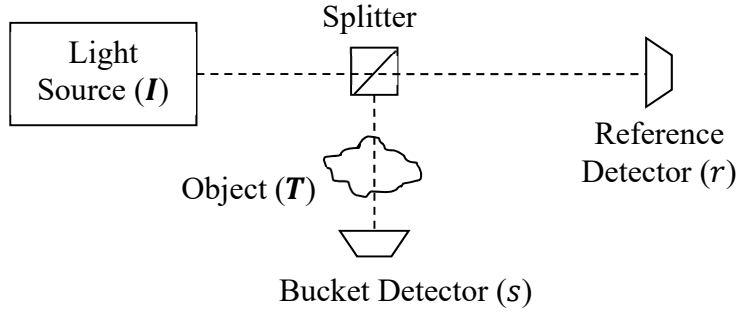
$$\mathbf{O} = \langle (I - \langle I \rangle) \left( s - \frac{\langle s \rangle}{\langle r \rangle} r \right) \rangle. \quad (14)$$

Alternatively, in the form of Eq. 9, DGI reconstruction is given by

$$\tilde{\mathbf{x}} = (\mathbf{A} - \text{vec}(\langle I \rangle) \mathbf{1}^T)^T \left( \mathbf{y} - \frac{\langle s \rangle}{\langle r \rangle} \mathbf{r} \right), \quad (15)$$

where

$$\mathbf{r} = (r_1, r_2, \dots, r_M)^T. \quad (16)$$



**Fig. 8 Diagram of DGI**

Another CGI variant is normalized GI (NGI).<sup>11</sup> NGI also makes use of a reference bucket measurement and is calculated as

$$\mathbf{O} = \langle (I - \langle I \rangle) \left( \frac{s}{r} - \frac{\langle s \rangle}{\langle r \rangle} \right) \rangle, \quad (17)$$

or in matrix form as

$$\tilde{\mathbf{x}} = (\mathbf{A} - \text{vec}(\langle I \rangle) \mathbf{1}^T)^T \left( \mathbf{y} \odot / \mathbf{r} - \frac{\langle s \rangle}{\langle r \rangle} \right), \quad (18)$$

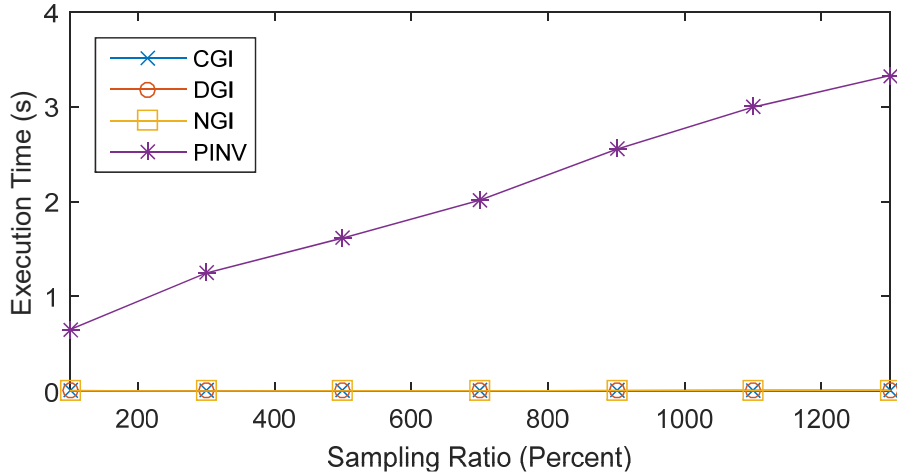
where  $\odot /$  denotes an element-wise division.

Examining the measurement model in Eq. 5, yet another reconstruction method presents itself. Given that GI can be modeled as a linear set of equations, given  $\mathbf{s}$  and  $\mathbf{A}$ , a vectorized estimation of the image  $\tilde{\mathbf{x}}$  can be calculated by

$$\tilde{\mathbf{x}} = \mathbf{A}^+ \mathbf{s}, \quad (19)$$

where  $\mathbf{A}^+$  is the pseudo-inverse (PINV) of  $\mathbf{A}$ .<sup>12</sup>

Figure 9 shows simulation results comparing CGI, DGI, NGI, and the PINV method for a range of sampling ratios. Once again  $\mathbf{I}$  is binary, and no noise sources were added to the simulation. DGI and NGI perform roughly the same but are both much better than CGI, with a 6-dB improvement for a sampling ratio of 1,500%. PINV is the best performer, especially at higher sampling ratios. Since the system is modeled by Eq. 5, there is perfect reconstruction when the sampling ratio reaches 100% and the PINV becomes a true inverse. Although in this case PINV performs better than the other reconstruction algorithms, it has a much longer execution time. Figure 9 shows the execution time the various GI reconstruction algorithms versus the sampling ratio, which is related to the size of  $\mathbf{A}$ . The algorithms were performed using MATLAB on a PC with an Intel Core i7-2760QM CPU running at 2.4 GHz and 8 GB of RAM. All the other algorithms take virtually no time compared with the much slower PINV method. In addition, the other algorithms can run as the measurements are acquired, while PINV can only be applied at the end of the measurement process.



**Fig. 9 Execution time of GI reconstruction algorithms vs. sampling ratio**

Although it was already demonstrated that DGI performs better than CGI, the true advantages of DGI and NGI can be seen when  $\tilde{\mathbf{I}}$  and  $\mathbf{I}'$  do not equal  $\mathbf{I}$ . Figures 10 and 11 are the same, except that the scale factors in Eq. 10 have been used in Fig. 11 to distort  $\tilde{\mathbf{I}}$  and  $\mathbf{I}'$  without any other additive noise; specifically,  $c = -0.2$  and  $b = 0.2/M$ . The performance of DGI and NGI, using the extra reference measurement, is comparable to the nonscaling case. CGI and PINV, which do not make use of the reference measurement, perform poorly when the illumination is distorted. Figure 12 summarizes the results of the nonscaled and scaled simulation in Figs. 10 and 11.

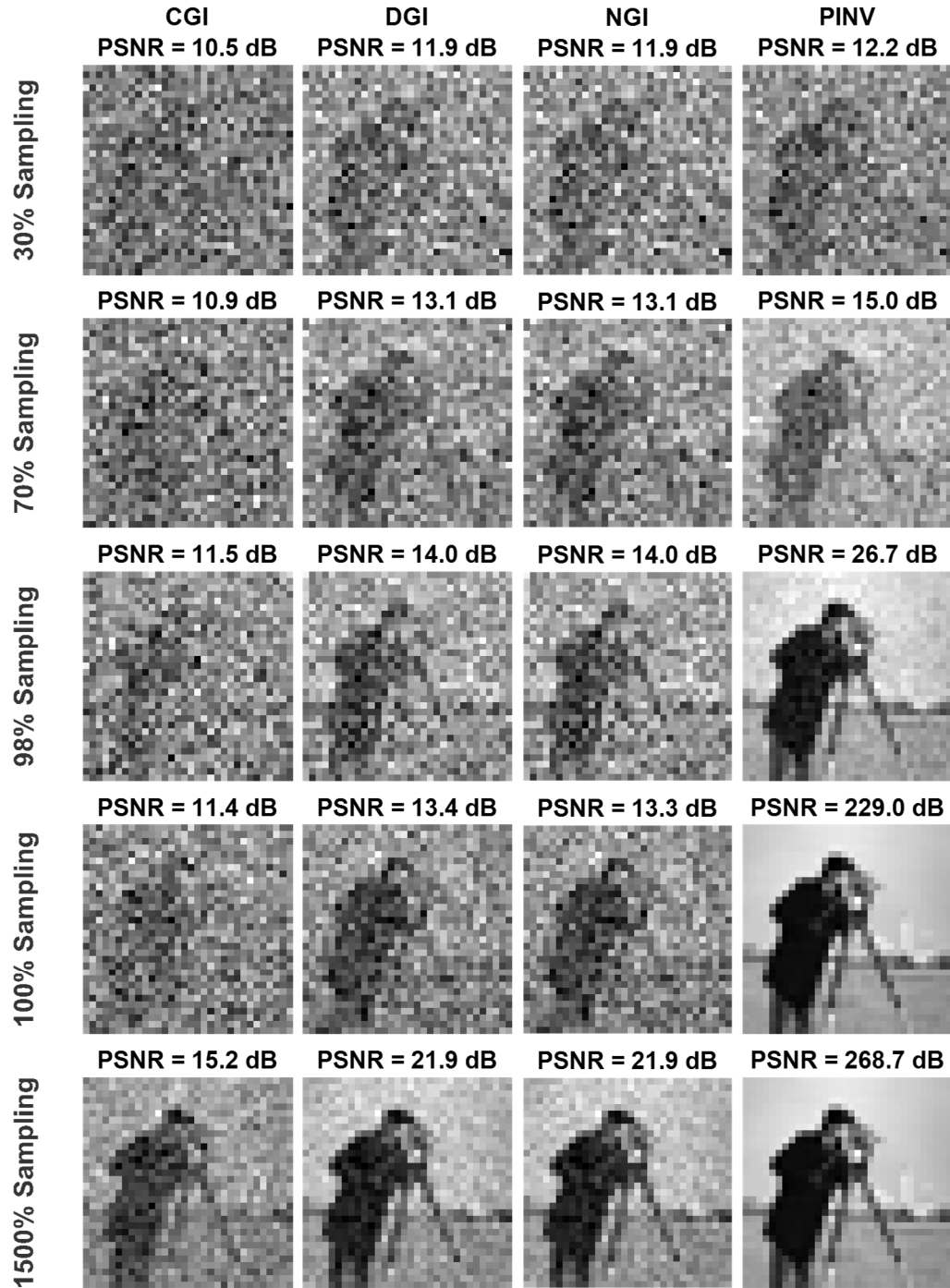
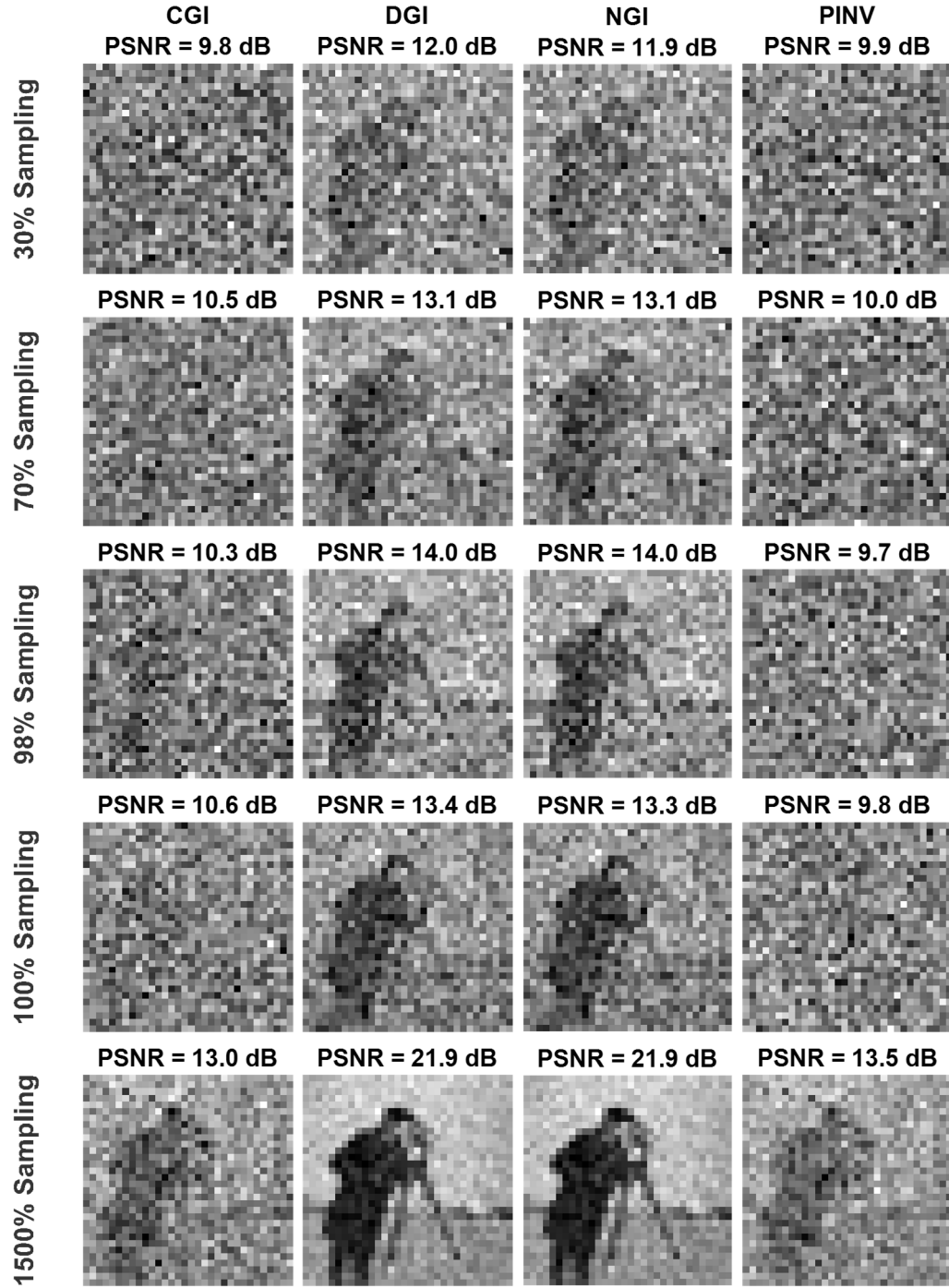
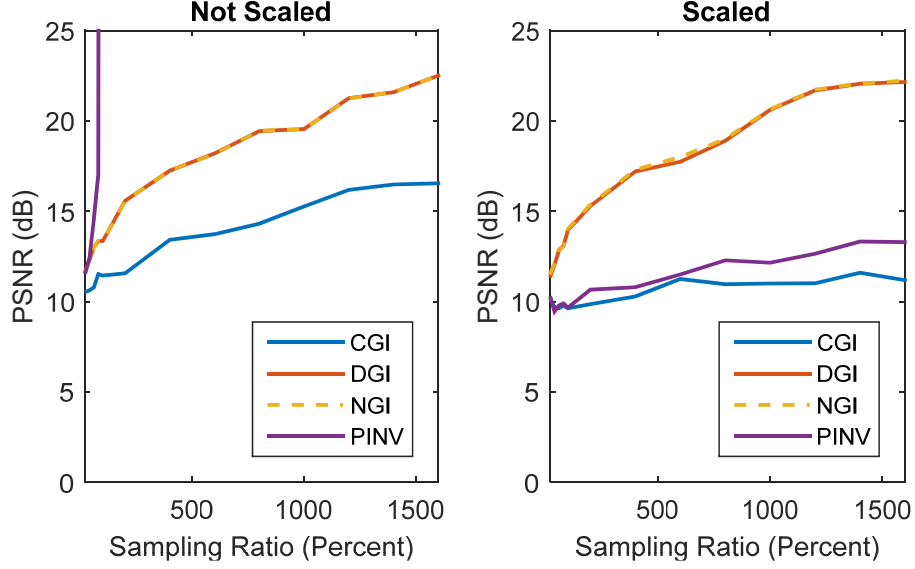


Fig. 10 Comparison of CGI, DGI, NGI, and PINV for several sampling ratios





**Fig. 11 Comparison of CGI, DGI, NGI, and PINV with scaled illumination for several sampling ratios**



**Fig. 12** Summary of the nonscaled and scaled simulation results for the simulations in Figs. 10 and 11

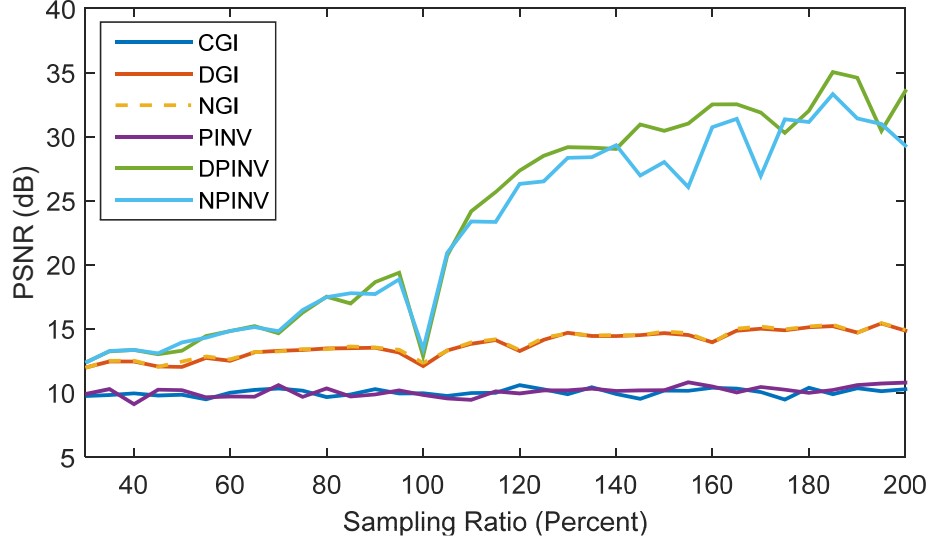
To improve performance in the case of distorted illumination, the reference measurement can be added the PINV. Differential PINV (DPINV) is formed by using the PINV in Eq. 15 to give

$$\tilde{\mathbf{x}} = \mathbf{A}^+ \left( \mathbf{y} - \frac{\langle s \rangle}{\langle r \rangle} \mathbf{r} \right). \quad (20)$$

Similarly, a normalized PINV (NPINV) is formed from Eq. 18 as

$$\tilde{\mathbf{x}} = \mathbf{A}^+ \left( \mathbf{y} \odot / \mathbf{r} - \frac{\langle s \rangle}{\langle r \rangle} \right). \quad (21)$$

Figure 13 shows the performance of DPINV and NPINV compared with the other GI variants for the scaled illumination case. The extra information in the reference measurement now allows DPINV and NPINV to outperform DGI and NGI.

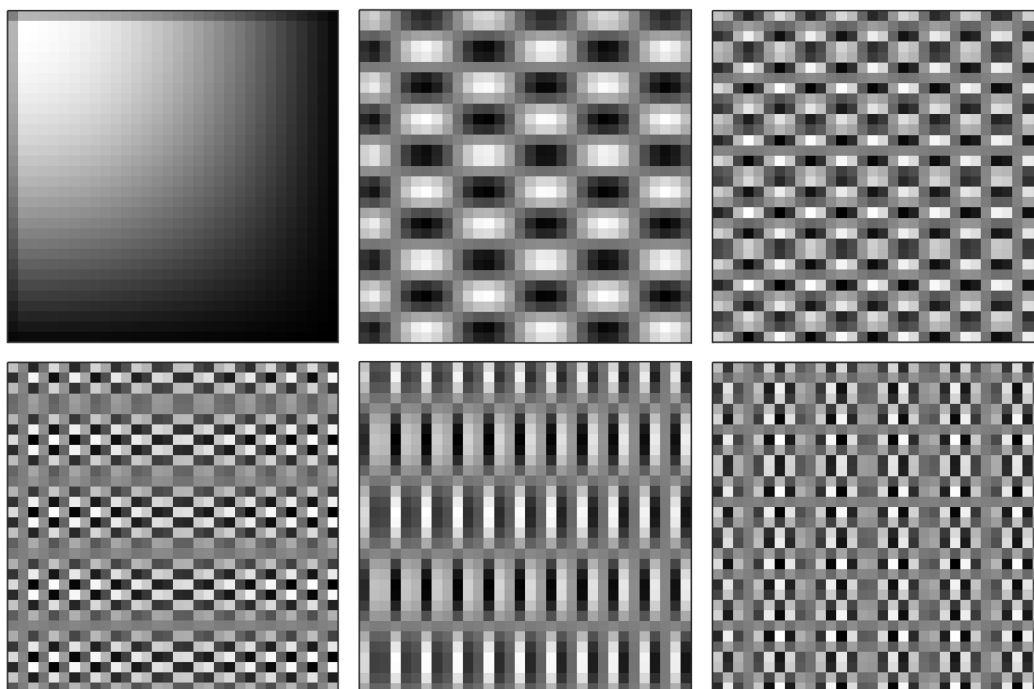


**Fig. 13** Performance comparison using scaled illumination including the new DPINV and NPINV algorithms

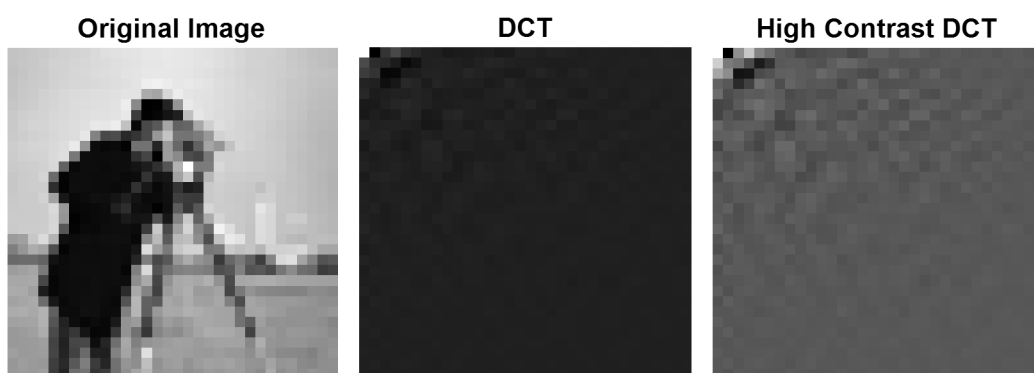
#### 4. Structured Illumination

The results so far have been obtained using random binary illumination patterns. Higher performance can be obtained, however, using pseudorandom or structured illumination.<sup>13</sup> Three patterns are considered: Fourier, Hadamard, and noiselet.

Fourier illumination uses the discrete cosine transform (DCT), a real-valued Fourier-related transform. Figure 14 shows some example DCT illumination patterns. DCT illumination can take advantage of the fact that natural images usually have most of their energy at lower frequencies. Figure 15 shows an example image with its DCT and a high-contrast DCT for illustration. The upper left corner of the DCT corresponds to lower frequencies, with the frequencies increasing toward the lower right corner. The largest DCT coefficients are in the upper left corner, containing most of the image's energy. Figure 16 shows another visualization demonstrating how the image's energy is concentrated in a small number of DCT coefficients. The 1024 pixel values of the  $32 \times 32$  original image were sorted in descending order and plotted as the standard basis. The DCT coefficients were similarly ordered and plotted. There are a relatively small number of large DCT coefficients compared with the number of large values in the standard basis. This means that most of the frequency information is concentrated in these few coefficients, allowing for a fairly accurate reconstruction of the image from a small number of DCT illumination patterns. Coefficients from a Walsh–Hadamard Transform (WHT), which is based on a Hadamard matrix, are also plotted in Fig. 16. Here too, most of the image's energy is concentrated in a small number of coefficients.



**Fig. 14** Example DCT illumination patterns



**Fig. 15** Example image, the DCT of the image, and a high-contrast visualization of the DCT. The lower frequencies of the DCT occur in the upper left and generally have the largest amplitudes.



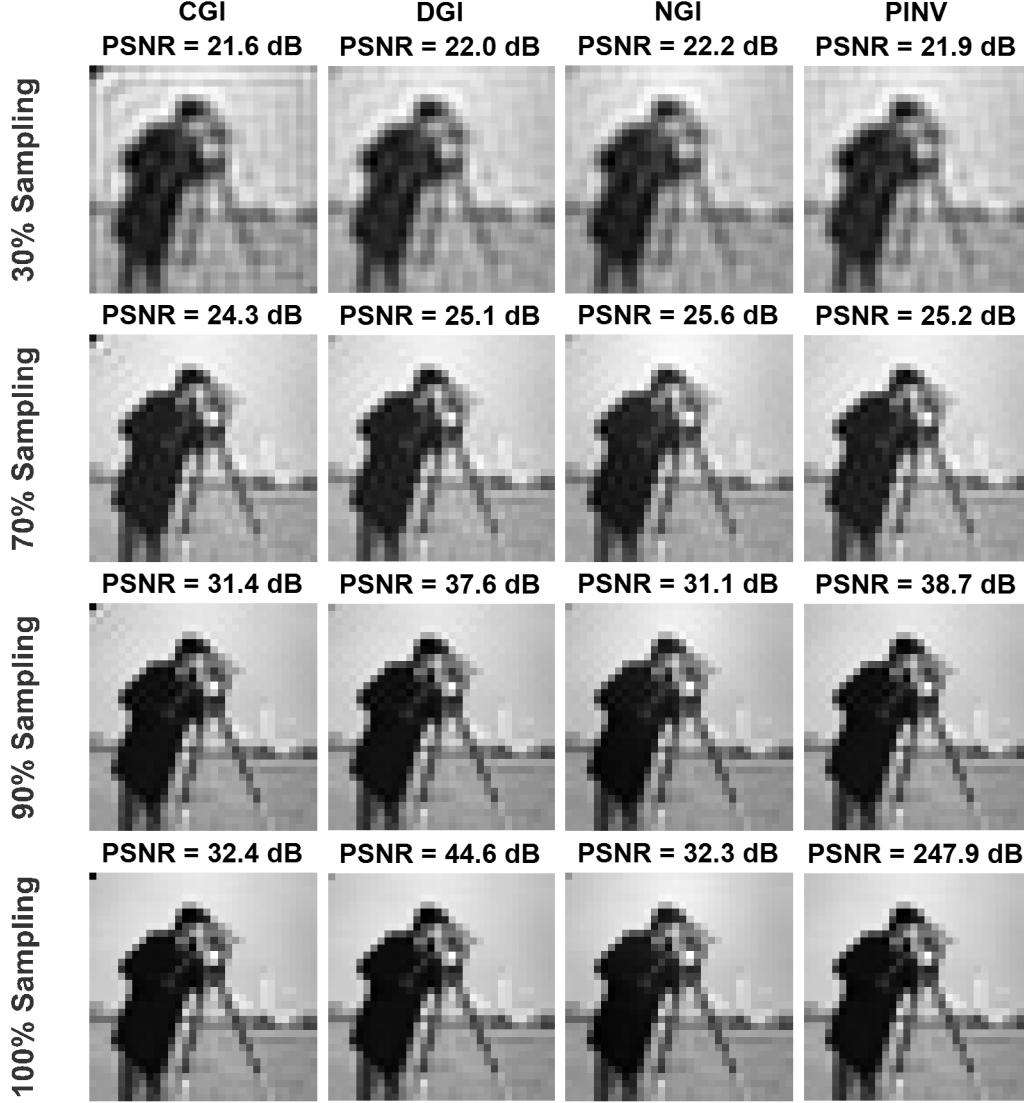


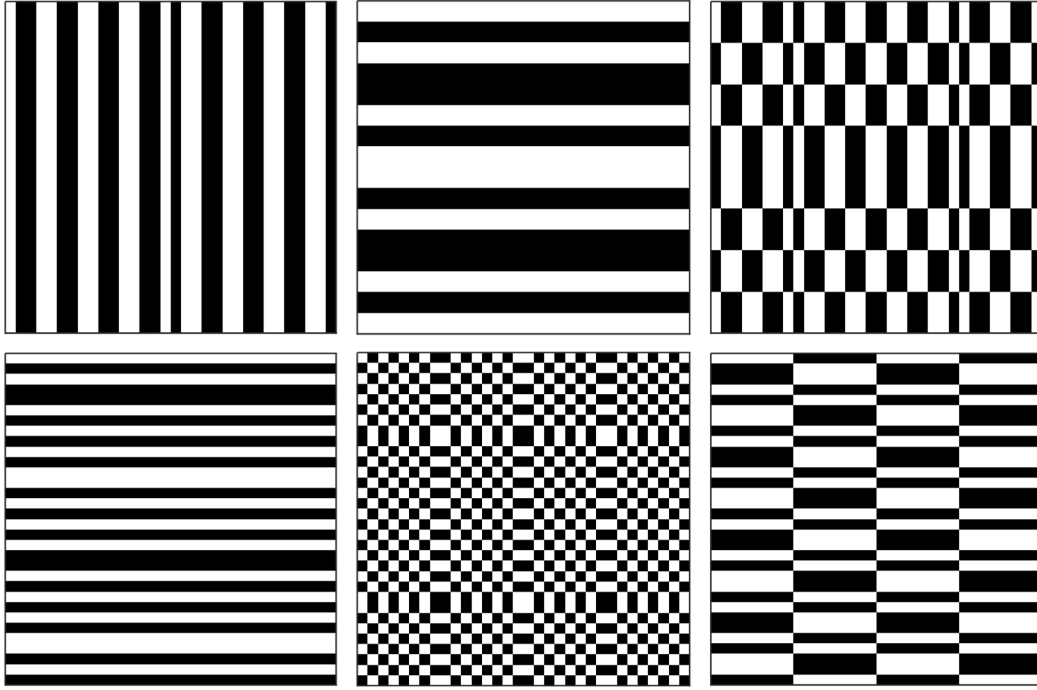
Fig. 18 Comparison of CGI, DGI, and PINV using DCT illumination for several sampling ratios

A Hadamard matrix<sup>15</sup>  $\mathbf{H}_k \in \mathbb{R}^{2^k \times 2^k}$  is defined recursively as  $\mathbf{H}_0 = 1$ , and

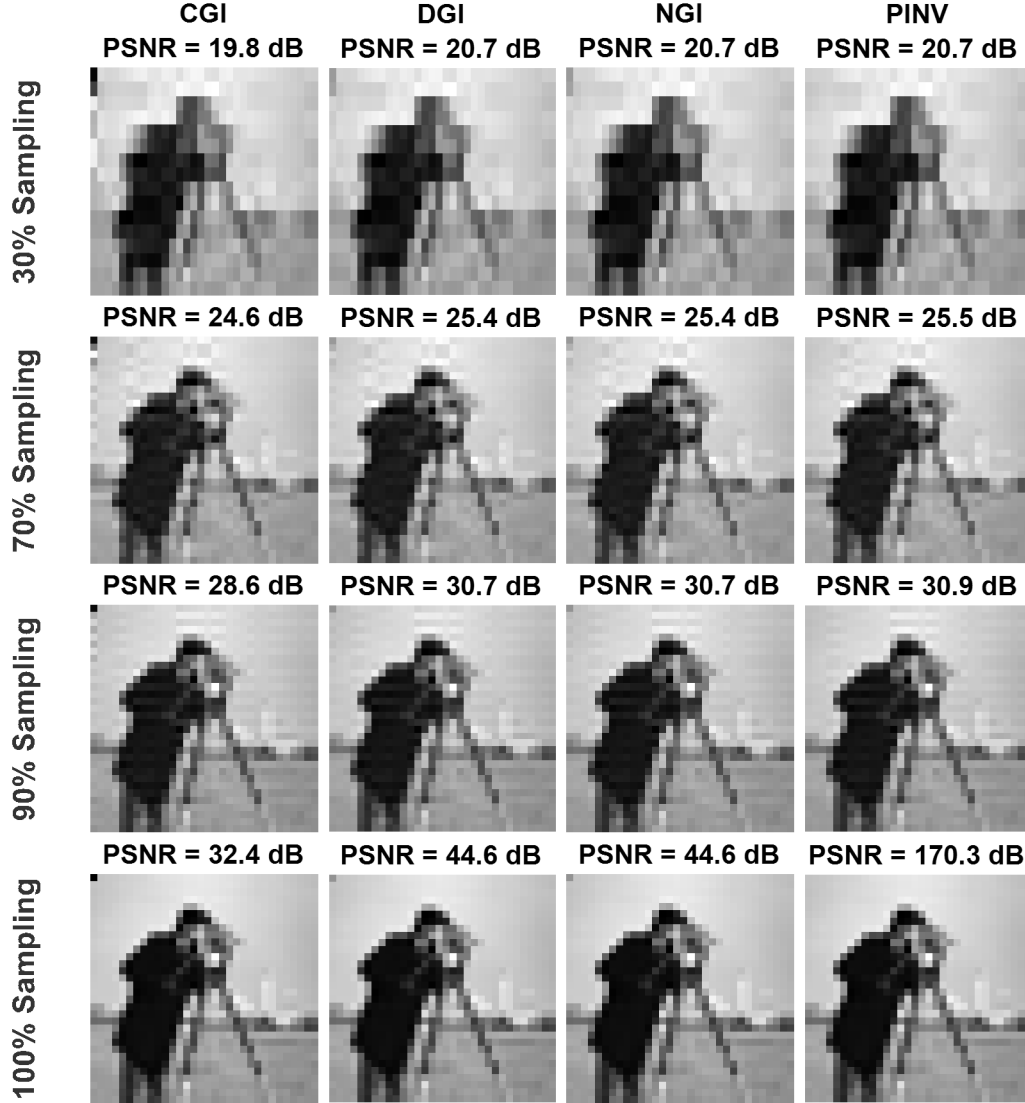
$$\mathbf{H}_k = \begin{bmatrix} \mathbf{H}_{k-1} & \mathbf{H}_{k-1} \\ \mathbf{H}_{k-1} & -\mathbf{H}_{k-1} \end{bmatrix}. \quad (22)$$

All of the elements are 1 or  $-1$ , and all of the rows are orthogonal. For GI with  $M$  measurements,  $M$  rows of a Hadamard matrix can be used as the sensing matrix  $\mathbf{A}$ . Negative values are changed to zero for illumination and can be treated as zero as well in the image recovery algorithm. Alternatively, recovery can be handled by first making a measurement under full illumination,  $s_0$ , and then calculating other measurements as  $s'_i = 2s_i - s_0$ . Figure 19 shows six example rows of a  $\mathbf{H}_{10}$

Hadamard matrix, reshaped as square illumination patterns. Figure 20 shows a comparison of CGI, DGI, and PINV using Hadamard illumination, without noise or scaling. Sequency ordering was used instead of Hadamard ordering, which places lower-frequency patterns first, improving image reconstruction performance. Hadamard illumination is clearly superior to the random binary illumination in Fig. 10.



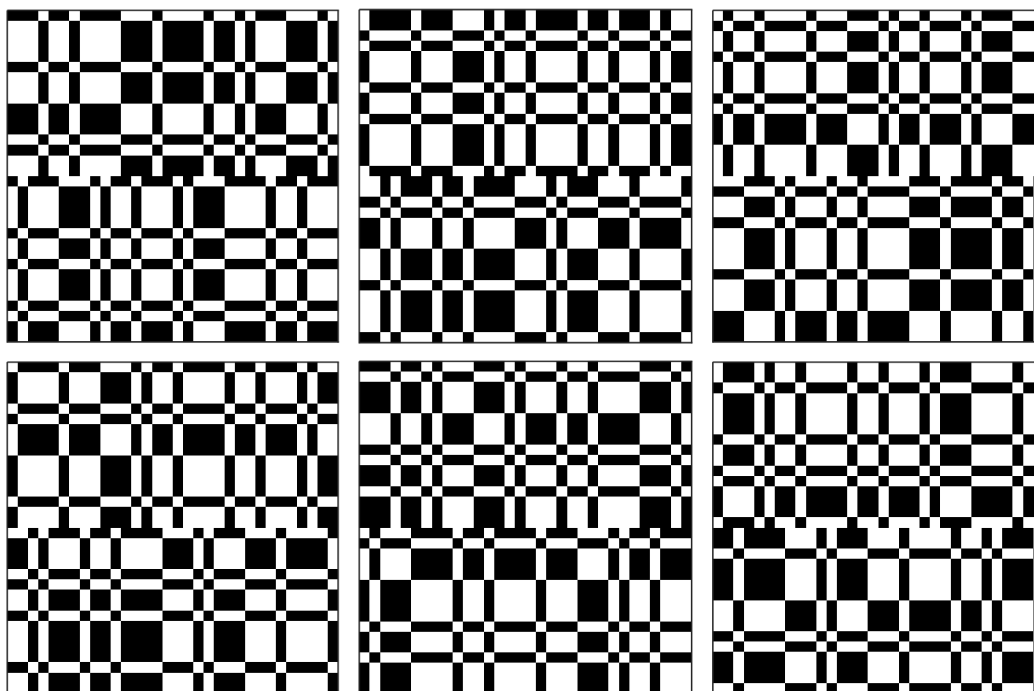
**Fig. 19** Example Hadamard illumination patterns



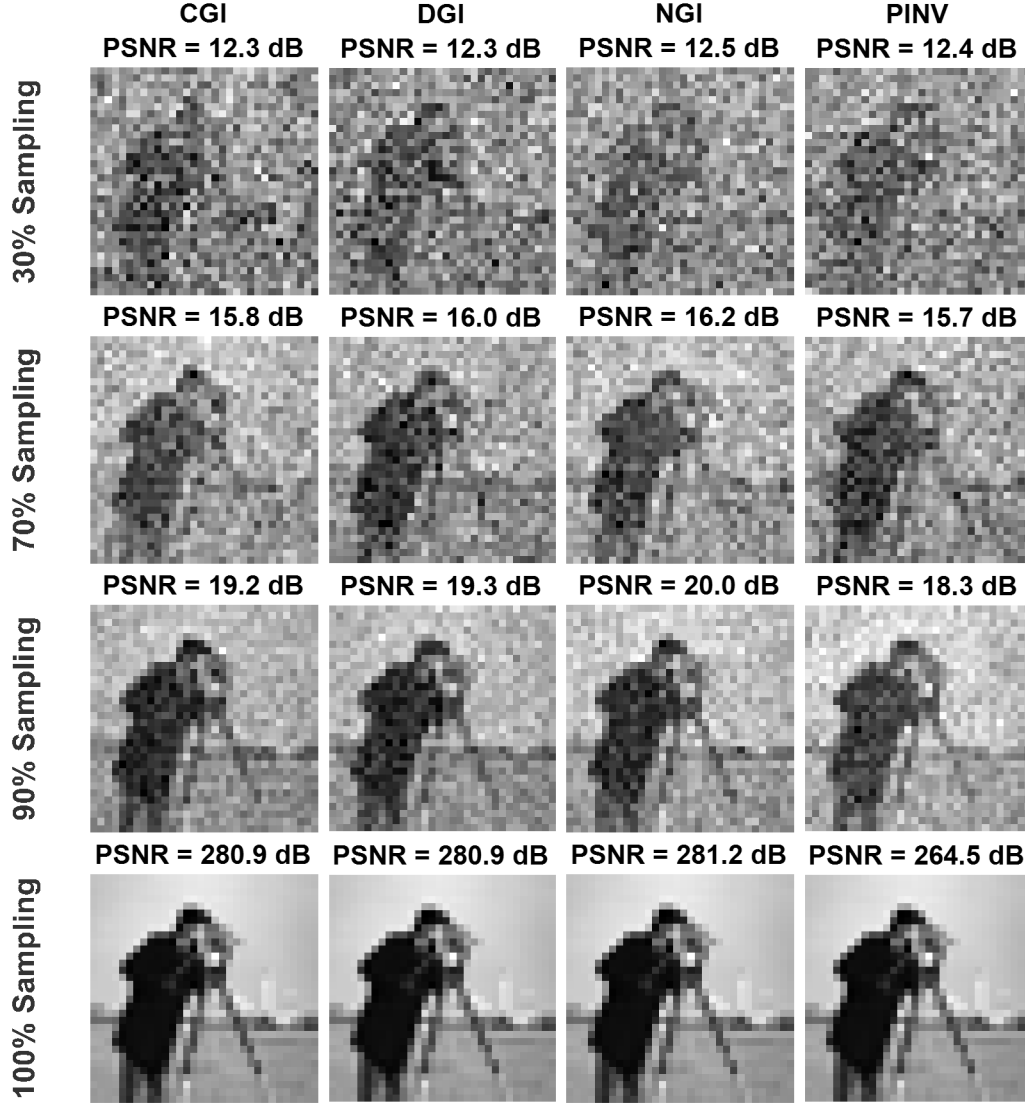
**Fig. 20** Comparison of CGI, DGI, NGI, and PINV using Hadamard illumination for several sampling ratios

Noiselets are pseudo-random binary functions that can be decomposed as a multiscale filterbank.<sup>16,17</sup> In general, they are complex-valued, although the implementation used here is real-valued. A real-valued dragon noiselet implementation is given in the function `realnoiseletm` in the Appendix, based on code by Justin Romberg.<sup>18</sup> Figure 21 shows six example illumination patterns generated by noiselets. Figure 22 shows a comparison of CGI, DGI, NGI, and PINV using noiselet illumination without noise or scaling. Noiselet illumination does not perform as well as Fourier and Hadamard illumination for most sampling ratios because the energy of natural images is not concentrated in a small number of coefficients in the noiselet basis. The patterns are orthogonal, however, leading to a near-perfect reconstruction at a 100% sampling ratio.



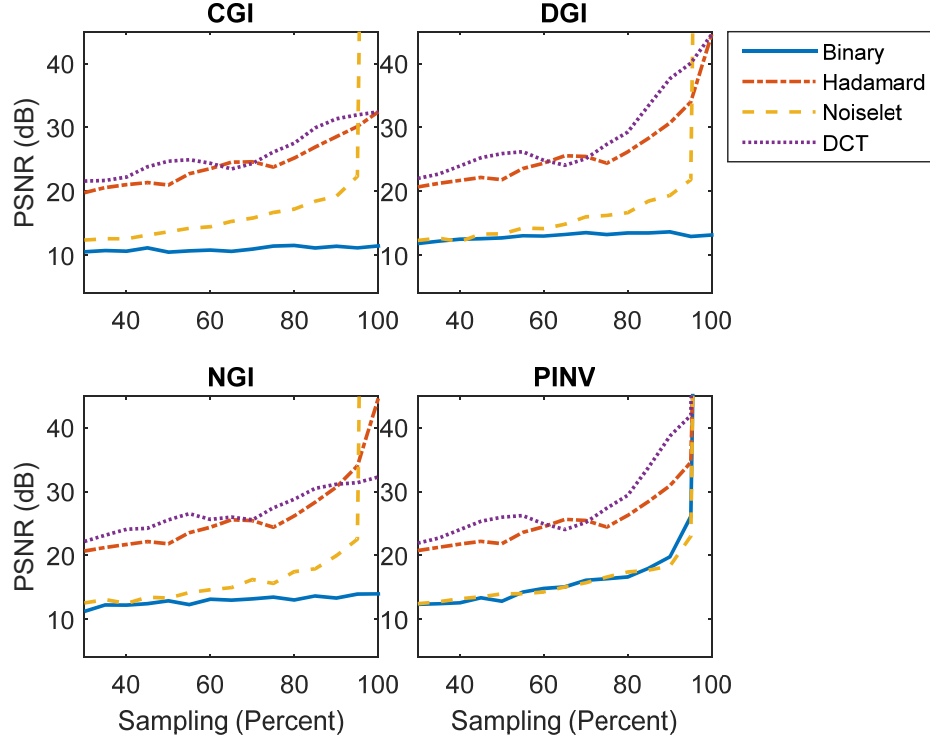


**Fig. 21**    **Example noiselet illumination patterns**

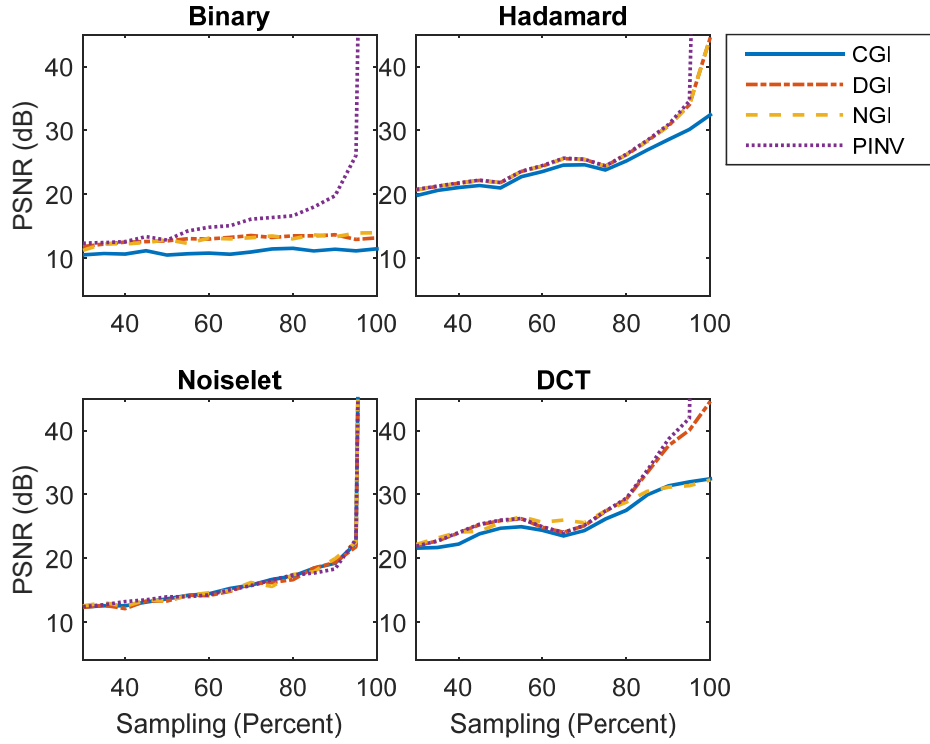


**Fig. 22 Comparison of CGI, DGI, NGI, and PINV using noiselet illumination for several sampling ratios**

Figures 23 and 24 summarize the simulation results of the various illumination patterns with different recovery algorithms. In general, DCT illumination performs the best across GI-recovery algorithms. DCT illumination is real-valued, however, and can be expensive to produce. In the case of binary-valued illumination, Hadamard performs almost as well as DCT. Comparing the GI algorithms for the various illumination types, PINV is the top performer. As mentioned previously, PINV is a time-consuming algorithm. Among the faster CGI variants, DGI is generally the next-best performer.



**Fig. 23** A performance summary comparing DCT, noiselet, Hadamard, and binary illumination using CGI, DGI, NGI, and PINV image recovery



**Fig. 24** Performance summary comparing CGI, DGI, NGI, and PINV image recovery algorithms for DCT, noiselet, Hadamard, and binary illumination

## 5. Denoising

Many noisy GI images, such as those in Fig. 10 and Fig. 21, can benefit from denoising.<sup>19</sup> Total variation (TV) regularization is explored here as a denoising algorithm, although many other denoising algorithms exist.<sup>20</sup> Following Gabriel Peyré's tutorial,<sup>21</sup> the TV of an image  $f$  is defined as

$$J(f) = \int \|\nabla f(x)\| dx. \quad (23)$$

Denoising is accomplished through minimizing a combination of a fit to the data  $x$  using the L2 norm and the weighted TV given by

$$\min_f \frac{1}{2} \|x - f\|^2 + \lambda J(f). \quad (24)$$

The minimization can be computed iteratively through gradient descent steps

$$f^{k+1} = f^k + \tau (f^k - x + \lambda \text{Grad}J(f^k)). \quad (25)$$

For  $f^k$  to converge,  $\tau$  should be set such that

$$\tau < \frac{2}{1 + \lambda \max_f \|D^2 J(f)\|}, \quad (26)$$

assuming that  $J$  is twice differentiable. In practice, the smooth TV norm

$$J_\varepsilon(f) = \int \sqrt{\varepsilon^2 + \|\nabla f(x)\|^2} dx \quad (27)$$

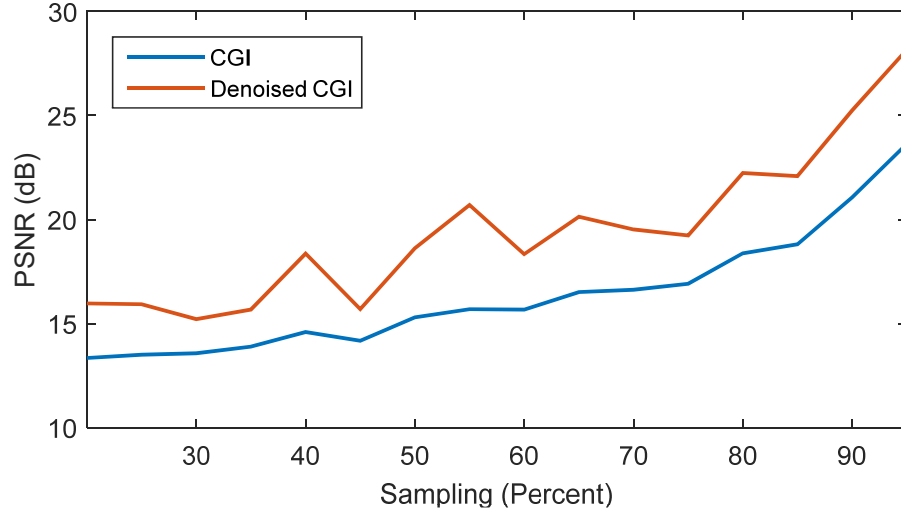
is used to make the gradient well defined. Its gradient is

$$\text{Grad}J_\varepsilon(f) = \text{div} \left( \frac{\nabla f}{\sqrt{\varepsilon^2 + \|\nabla f\|^2}} \right). \quad (28)$$

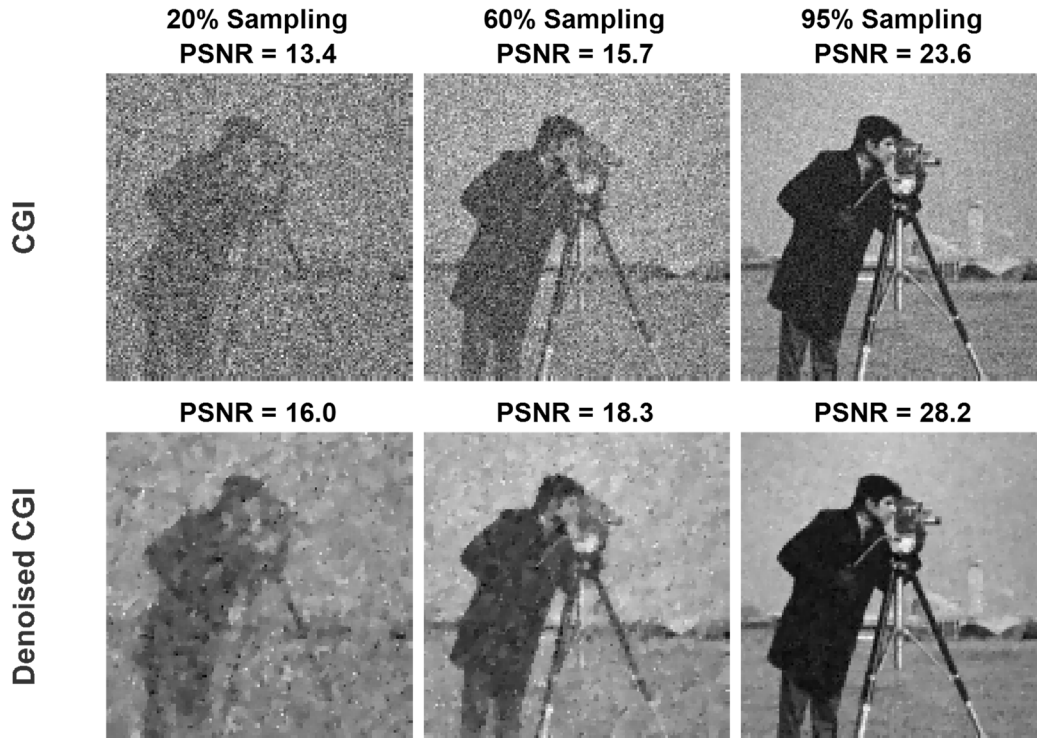
Figures 25 and 26 show the results of denoising 128×128 CGI recovered images using noiselet illumination. The best result of 800 iterations was chosen as the denoised image, using  $\lambda = 0.1$ ,  $\varepsilon = 0.01$ , and

$$\tau = \frac{2}{1 + \lambda 8/\varepsilon}. \quad (29)$$

In this case, the denoising simulations in Fig. 25 resulted in an average 3-dB gain in PSNR.



**Fig. 25** Performance gain of denoising an image recovered through CGI over a range of sampling ratios



**Fig. 26** Examples of denoising CGI images for 3 sampling ratios

## 6. Compressive Sensing

CS is a relatively recent sensing technique where, unlike traditional sampling, the number of measurements required depends on the sparsity of the signal instead of its bandwidth. There are many tutorials that introduce CS,<sup>17,22–24</sup> and the CCDC

Army Research Laboratory has researched the benefits of applying CS to a number of different sensing problems.<sup>25–29</sup> The sensing procedure of CS is similar to CGI in many applications; therefore, it is natural to apply CS to GI.<sup>30</sup> Like GI, CS can be modeled as patterned illumination collected in a bucket detector, leading to the same sensing model as in Eq. 5,  $\mathbf{s} = \mathbf{A}\mathbf{x}$ . CS requires the image  $\mathbf{x}$  to be sparse in some basis, a quality possessed by most natural images, as previously illustrated in Fig. 15 and 20. Given the sparse coefficients  $\boldsymbol{\theta}$  of  $\mathbf{x}$  in basis  $\boldsymbol{\Psi}$ , the measurements  $\mathbf{s}$  can be rewritten as

$$\mathbf{s} = \mathbf{A}\boldsymbol{\Psi}\boldsymbol{\theta} = \mathbf{G}\boldsymbol{\theta}, \quad (30)$$

where  $\mathbf{x} = \boldsymbol{\Psi}\boldsymbol{\theta}$  and the system matrix  $\mathbf{G}$  is defined as  $\mathbf{G} = \mathbf{A}\boldsymbol{\Psi}$ . Given that  $N > M$ , there are an infinite number of solutions to Eq. 30, but CS theory states that if there are a sufficient number of measurements, the sparsest solution will recover the original signal. This can be found through L1-norm minimization denoted as

$$\min_{\boldsymbol{\theta}} \|\boldsymbol{\theta}\|_1 \quad \text{subject to } \mathbf{s} = \mathbf{G}\boldsymbol{\theta}. \quad (31)$$

The number of measurements required to reconstruct  $\mathbf{x}$  is

$$M \geq C \cdot \mu^2 \cdot \|\boldsymbol{\theta}\|_0 \cdot \log N, \quad (32)$$

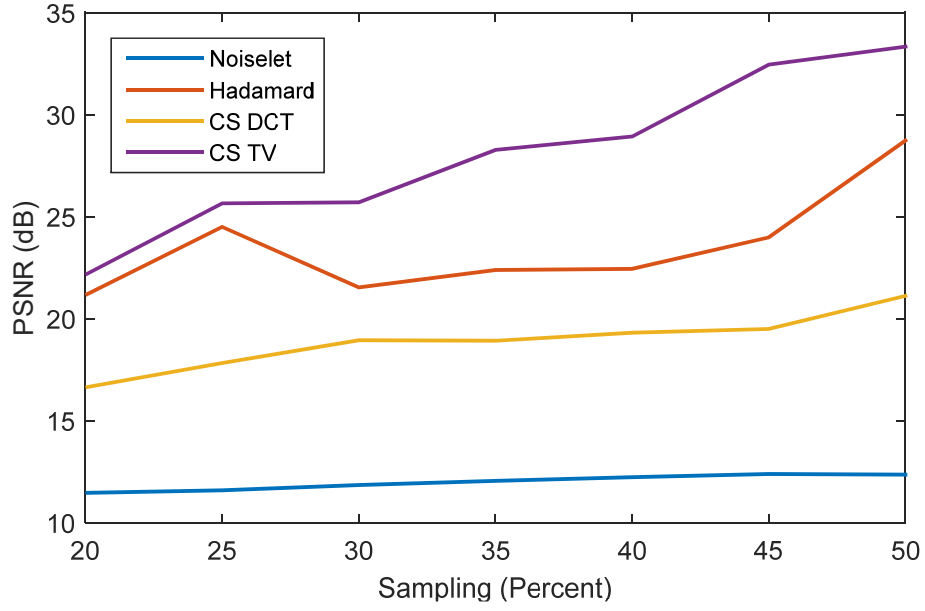
where  $C$  is a positive constant,  $\|\boldsymbol{\theta}\|_0$  is the number of nonzero values in  $\boldsymbol{\theta}$ , and  $\mu$  is a small constant determined by the structure of  $\mathbf{G}$  called mutual coherence. The important point is that the number of measurements is linearly related to its sparsity  $\|\boldsymbol{\theta}\|_0$ , but only logarithmically related to the signal's size, allowing large sparse signals to be sampled with relatively few measurements.

In practice, TV minimization given by

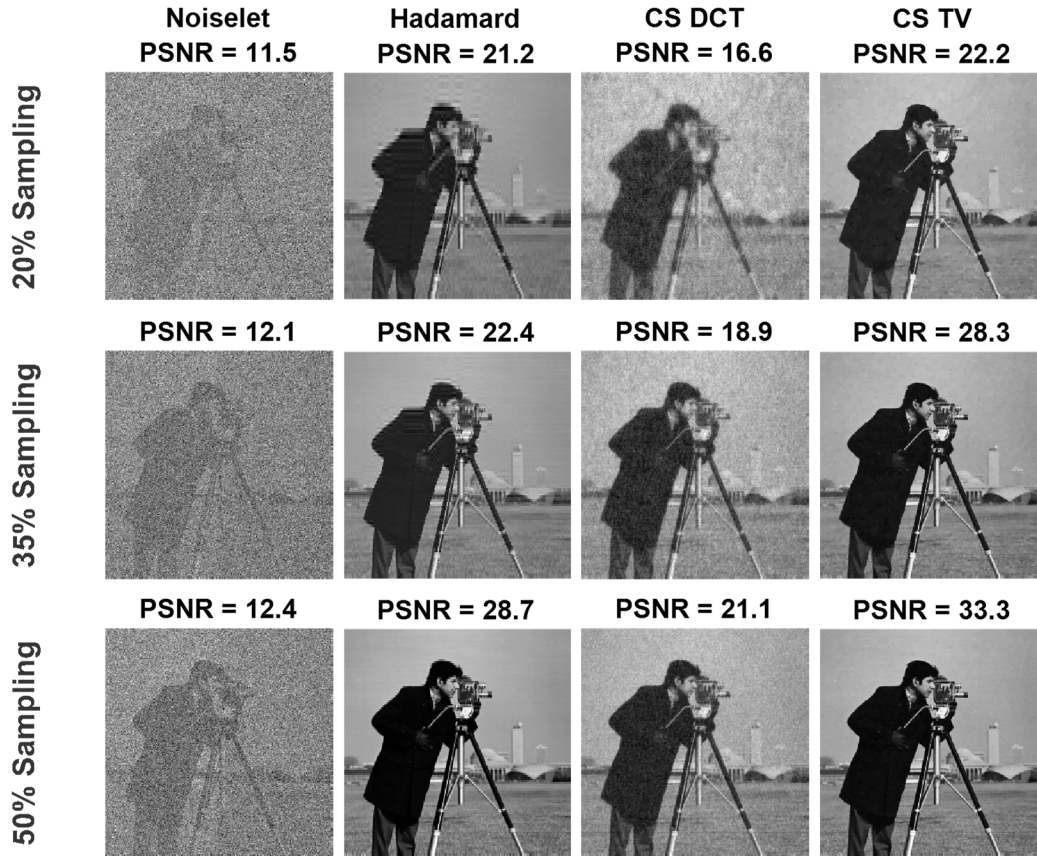
$$\min_{\mathbf{x}} J(\mathbf{x}) \quad \text{subject to } \mathbf{s} = \mathbf{A}\mathbf{x} \quad (33)$$

also works well, since the gradient of natural images is sparse.  $J$  here is a discrete version of the TV norm defined in Eq. 23.

Figures 27 and 28 show a comparison of CGI using noiselet illumination, Hadamard illumination, CS using a DCT basis, and CS using TV minimization on an example  $256 \times 256$  image. The CS results were obtained from the open-source L1 Magic software package,<sup>31</sup> using noiselets to generate the sensing matrix. CS using TV minimization is the best performer, but GI also does well, outperforming CS using the DCT basis in this case.

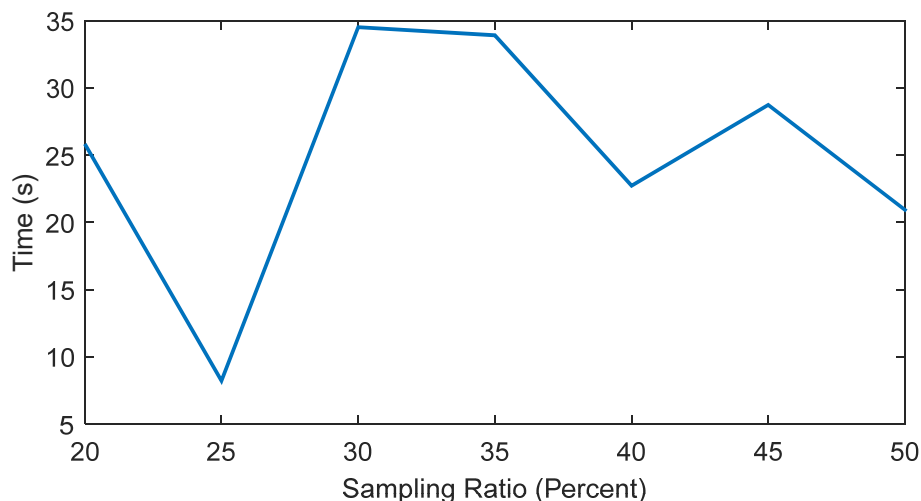


**Fig. 27** Comparison of CGI with noiselet and Hadamard illumination, and CS using a DCT basis and TV minimization



**Fig. 28** Example images comparing CGI with noiselet and Hadamard illumination, and CS using a DCT basis and TV minimization

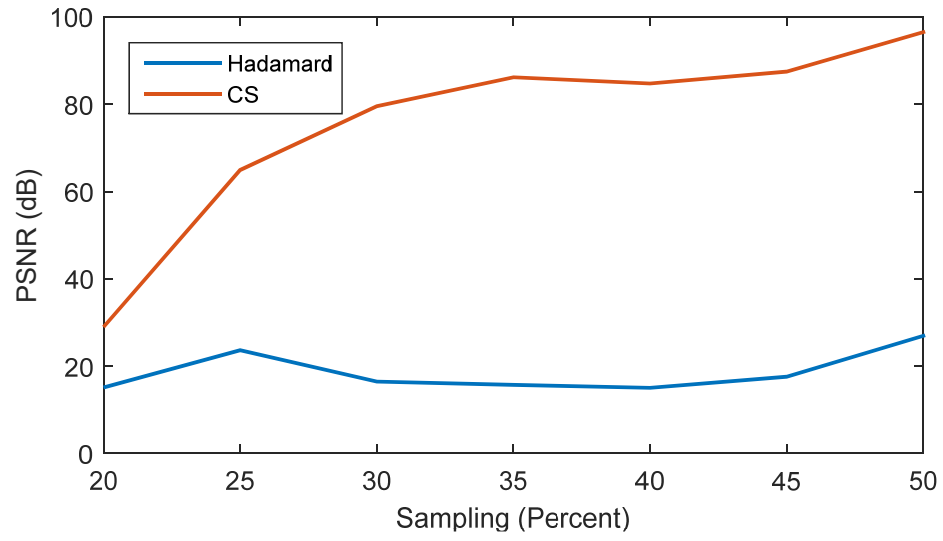
There are three important points regarding these results. First, CS-minimization algorithms are computationally intensive and have a long execution time compared with GI. Figure 29 shows the execution time of the CS simulations used to produce the CS DCT results in Fig. 27. This is much longer than the CGI noiselet algorithm, which executed in about 2 ms. These results were obtained using the same PC as for Fig. 9. Not only does CS have a long execution time, but it can only begin once all of the measurements have been made. GI algorithms can be performed as each measurement is acquired. Second, noiselets do not enhance CS performance much over a random binary pattern. This means that in the case of random illumination, CS will greatly outperform TGI, similar to the noiselet-illuminated CGI and CS simulations presented here. Third, the performance of CS depends on the sparsity of the image. Although most natural 2-D images are sparse, they are only modestly sparse compared with many 3-D data applications, such as magnetic resonance imaging, hyper-spectral imaging, and 3-D antenna patterns. Thus, on very sparse images CS will perform extremely well, while the performance of GI will remain the same.



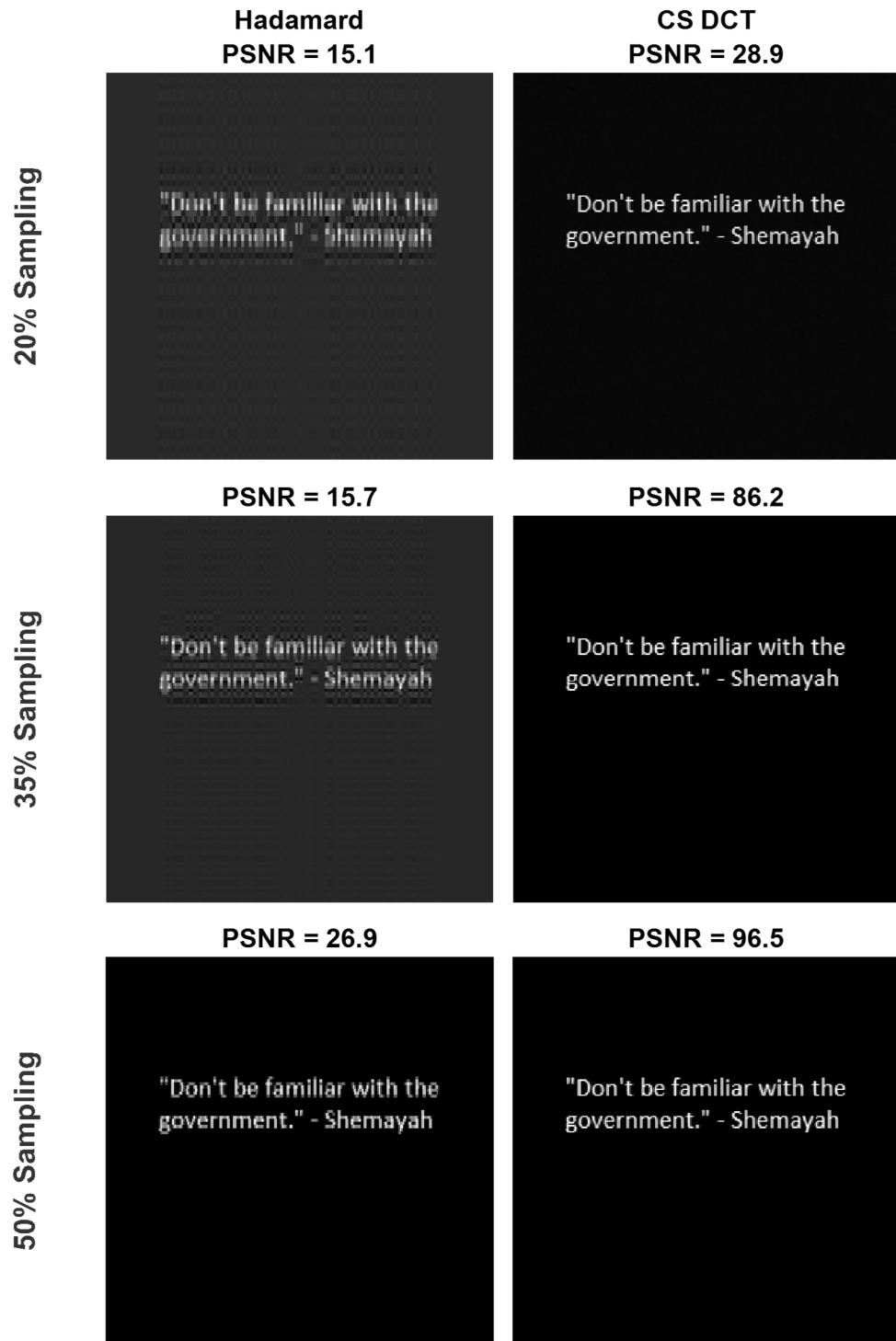
**Fig. 29 Execution time of the CS algorithm using the DCT basis**

This third point is demonstrated in Figs. 30 and 31, comparing the performance of Hadamard illuminated GI with CS in the standard basis. In this example, the image of text is extremely sparse, with only 2.6% of the pixel values nonzero. In this case, the performance difference between CS and GI is much larger than that of the cameraman test image used in Fig. 27. In Figs. 32 and 33 a new test image was formed by taking the inverse DCT of the text image used in Fig. 31, ensuring that it is sparse in the DCT basis. Here CS also performs much better than GI, now using CS with the DCT basis. The contrast of the images in Fig. 33 was increased for illustration purposes.

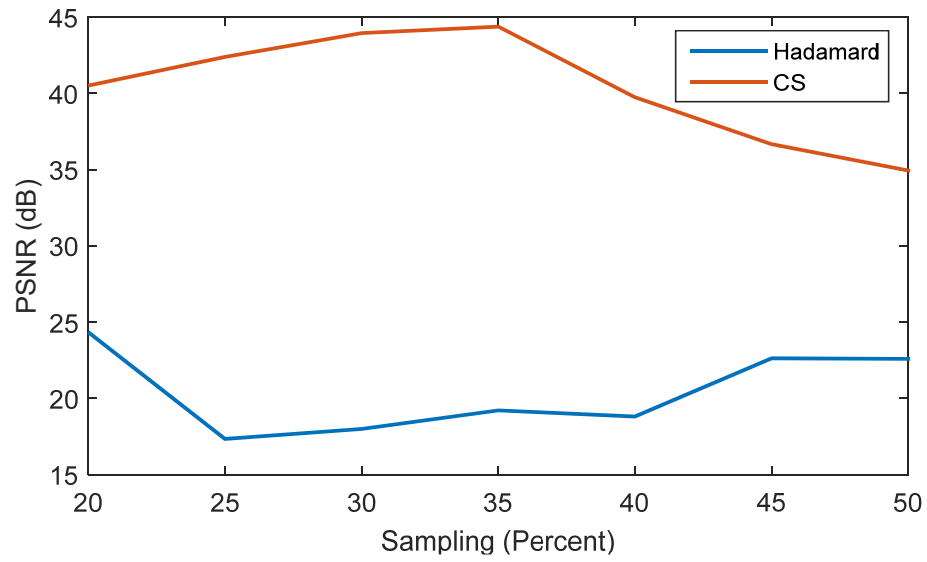




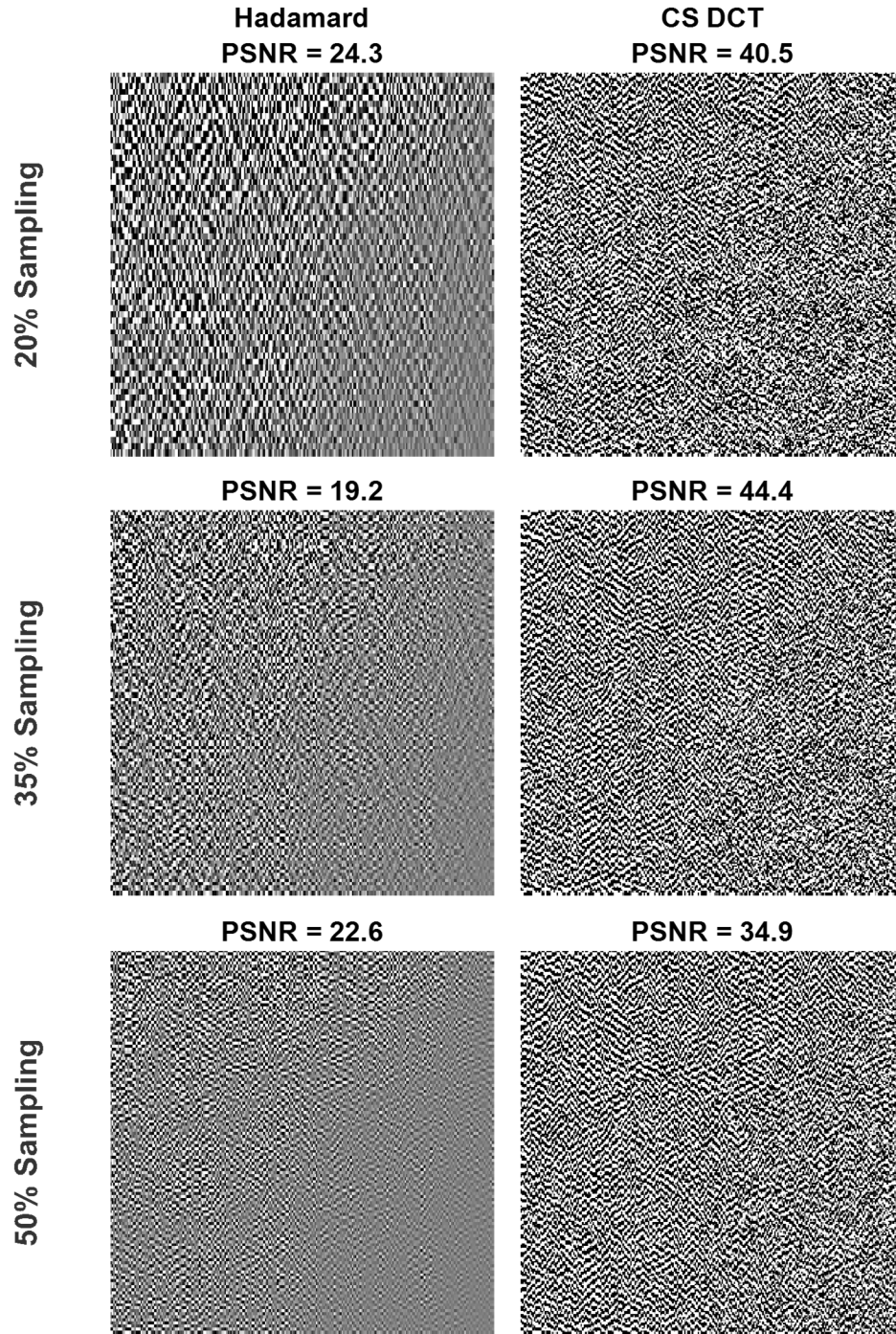
**Fig. 30** Comparison of GI and CS using the image sparse in the standard basis shown in Fig. 31



**Fig. 31** Example images comparing GI with CS using a sparse image in the standard basis



**Fig. 32** Comparison between GI and CS using the image sparse in the DCT basis shown in Fig. 33



**Fig. 33** Example high-contrast images comparing GI with CS using a sparse image in the DCT basis

## 7. Conclusion

---

This report introduced GI, comparing several different GI algorithms. When the illumination source is random, a high-resolution FPA must be used to record the illumination patterns. In these cases, CS reconstruction typically produces much-higher-quality images than GI, although their reconstruction algorithms are much more computationally intensive. If the illumination source is controlled, GI performance can be enhanced using structured patterns such as Hadamard or DCT illumination. In these cases it is possible to dispense with the FPA, although often a second-reference bucket detector will aid image reconstruction using DGI or NGI algorithms. The quality of CGI compared with TGI with an FPA will depend on how accurately the real illumination matches the programmed illumination. CS reconstruction may also outperform GI algorithms for structured illumination, although the performance gain will be less than that seen for random illumination. In addition to the reconstruction algorithm, GI can benefit from denoising during postprocessing. Example MATLAB code is provided in the Appendix that implements many of the algorithms presented here, allowing others to easily begin their own exploration into GI.

## 8. References

---

1. Erkmen BI, Shapiro JH. Ghost imaging: from quantum to classical to computational. *Advances in Optics and Photonics*. 2010;2(4):405–450.
2. Pittman TB, Shih YH, Strekalov DV, Sergienko AV. Optical imaging by means of two-photon quantum entanglement *Phys Rev A*. 1995;52(5).
3. Brambilla Gatti E, Bache M, Lugiato LA. Ghost imaging with thermal light: comparing entanglement and classical correlation. *Phys Rev Lett*. 2004;93:093602.
4. Shapiro JH. Computational ghost imaging. *Phys Rev A*. 2008;78.6:061802.
5. Bromberg Y, Katz O, Silberberg Y. Ghost imaging with a single detector. *Phys Rev A*. 2009;79:053840.
6. Basano L, Ottonello P. Ghost imaging: open secrets and puzzles for undergraduates. *American Journal of Physics*. 2007;75(4):343–351.
7. Padgett MJ, Boyd RW. An introduction to ghost imaging: quantum and classical. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*. 2017;375(2099):20160233.
8. Hoenders BJ. Review of a bewildering classical–quantum phenomenon: ghost imaging. *Advances in Imaging and Electron Physics*. 2018;208:1.
9. Pal D. Review of ghost imaging; 2015 Aug [accessed 2019 Jan 7]. [https://www.researchgate.net/publication/281103553\\_Review\\_of\\_Ghost\\_Imaging](https://www.researchgate.net/publication/281103553_Review_of_Ghost_Imaging).
10. Ferri F, Magatti D, Lugiato LA, Gatti A. Differential ghost imaging. *Physical Review Letters*. 2010;104(25):253603.
11. Sun Baoqing, Welsh SS, Edgar MP, Shapiro JH, Padgett MJ. Normalized ghost imaging. *Optics Express*. 2012;20(15):16892–16901.
12. Zhang C, Guo S, Cao J, Guan J, Gao F. Object reconstitution using pseudo-inverse for ghost imaging. *Optics Express*. 2014;22(24):30063–30073.
13. Zhang Z, Wang X, Zheng G, Zhong J. Hadamard single-pixel imaging versus Fourier single-pixel imaging. *Optics Express*. 2017;25:19619–19639.
14. Wallace GK. The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics*. 1992;38(1):xviii–xxxiv.

15. Hedayat A, Wallis WD.. Hadamard matrices and their applications. *The Annals of Statistics*. 1978;6(6):1184–1238.
16. Coifman Ronald, Geshwind F, Meyer Yves. Noiselets. *Applied and Computational Harmonic Analysis*. 2001;10.1:27–44.
17. Candes Emmanuel, Romberg Justin. Sparsity and incoherence in compressive sampling. *Inverse Problems*. 2007;23(3):969.
18. Romberg Justin. Imaging via compressive sampling: introduction to compressive sampling and recovery via convex programming. *IEEE Signal Processing Magazine*. 200;25(2):14–20.
19. Yao Xu-Ri, Yu Wen-Kai, Liu Xue-Feng, Li Long-Zhen, Li Ming-Fei, Wu Ling-An, Zhai Guang-Jie. Iterative denoising of ghost imaging. *Optics Express*. 2014;22(20):24268–24275.
20. Buades Antoni, Coll Bartomeu, Morel Jean-Michel. A review of image denoising algorithms, with a new one. *Multiscale Modeling & Simulation*. 2005;4(2):490–530.
21. Peyré G. The numerical tours of signal processing: advanced computational signal and image processing. *IEEE Computing in Science and Engineering*. 2011;13(4):94–97.
22. Candès Emmanuel J, Wakin Michael B. An introduction to compressive sampling. *IEEE Signal Processing Magazine*. 2008;25(2):21–30.
23. Duarte Marco F, Davenport Mark A, Takhar Dharmpal, Laska Jason N, Sun Ting, Kelly Kevin F, Baraniuk Richard G. Single-pixel imaging via compressive sampling. *IEEE Signal Processing Magazine*. 2008;25(2):83–91.
24. Willett Rebecca M, Marcia Roummel F, Nichols Jonathan M. Compressed sensing for practical optical imaging systems: a tutorial. *Optical Engineering*. 2011;50(7):072601.
25. Chen X, Yu Z, Hoyos S, Sadler BM, Silva-Martinez J. A sub-nyquist rate sampling receiver exploiting compressive sensing. *IEEE Transactions on Circuits and Systems I: Regular Papers*. 2011;58(3):507–520.
26. Kim HH, Govoni MA, Haimovich AM. May. Cost analysis of compressive sensing for MIMO STAP random arrays. *IEEE Radar Conference (RadarCon)*. 2015:0980–0985. doi: 10.1109/RADAR.2015.7131137.
27. Don ML. Fu C, Arce GR. Compressive imaging via a rotating coded aperture. *Applied Optics*. 2017;56(3):B142–B153.

28. Fu C, Don ML, Arce GR. Compressive spectral imaging via polar coded aperture. *IEEE Transactions on Computational Imaging*, 2017;3(3):408–420.
29. Don ML, Arce GR. Antenna radiation pattern compressive sensing. *MILCOM* 2018. doi: 10.1109/MILCOM.2018.8599791.
30. Katz Ori, Bromberg Yaron, Silberberg Yaron. Compressive ghost imaging. *Applied Physics Letters*. 2009;95.13:131110.
31. Candes Emmanuel, Romberg Justin. l1-magic: recovery of sparse signals via convex programming; 2005 Oct [accessed 2019 Nov 26]. <http://www-inst.eecs.berkeley.edu/~ee225b/fa12/lectures/CSmeetsML-Lecture1/codes/l1magic/l1magic.pdf>.



## **Appendix. Example MATLAB Code**

---

The example code consists of a main GI function in GI.m, and two test scripts, GI\_noise\_test.m and GI\_test.m. GI\_noise\_test.m will reproduce (from the main report) Fig. 7, and GI\_test.m will reproduce Fig. 20, Fig. 22, Fig. 18, Fig. 23, and Fig. 24. To use the “noiseletf” illumination in GI.m, you must download and install Justin Romberg’s noiselet code. The original code was unavailable at the time of publication, but was found on an alternative website.\* Further documentation of the example code is contained in the code comments.

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   GI.m: Main GI function   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function
[O,time,PSNR]=GI(img,m,gi_type,illum_type,scale,vs,vI,vI2,do_denoise)
%Inputs
% img - image (square, sides are power of 2)
% m - measurements
% gi_type - CGI DGI NGI PINV DPINV NPINV JPEG
% illum_type - 'hadamardf' 'hadamardm' 'binarym' 'realm' 'noiseletf'
%            'noiseletm' 'dctsm' 'dctzzm'
% Vnoise - variance of noise added to bucket and reference bucket
% scale - scale measurements linearly from 1 to 1+scale
% vs - noise standard dev added to bucket and reference bucket
% vI - noise added to illumination
% vI2 - noise added to measured illumination
% do_denoise - 1 = denoise, 0 = don't denoise
%Outputs
% O - recovered image (vectorized)
% time - processing time
% PSNR - in dB

%illum type notes:
%   end in m, matrix implementation
%   end if f, functional implementation

%If TGI or PINV variant, or use noise or scale, must use a matrix illum
%For large images, use functional illumination, don't use PINV variant

%Support
% noiseletf requires noiselet mex code by Justin Romberg
% The denoising and supporting functions are based on code by Gabriel

%Examples
%   [O,time,PSNR]=GI(rand(4,4),8,'TGI','hadamardm',1,0,0,0,0)
%   [O,time,PSNR]=GI(img,m,'TGI','binarym',scale,Vs,Vil,Vi2,0)

n=length(img(1,:));
img=imnormalize(img);
%each row of I is vec(illum. pattern)
switch illum_type
case 'hadamardm'
%       Im=hadamard(n^2); %hadamard ordering
%       Im=Im(1:m,:);
%       %Im=(Im+1)/2;
Iml=index(fwht(eye(n^2),n^2),1:m); %sequency ordering
Iml=(Iml+1)/2;
case 'hadamardf'

```

---

\*22. Romberg, J. Compressive Sensing [accessed 2019 Jan 7]. <http://w3.impa.br/~aschulz/CS/paper.html>.

```

I1 = @(z) index(fwht(z,n^2),1:m); %sequency ordering
It1 = @(z) ifwht(unindex(z,1:m,n^2));
case 'noiseletf'
q = randperm(n^2)';
I1 = @(z) index(realnoiselet(z),q(1:m));
It1 = @(z) realnoiselet(unindex(z, q(1:m), n^2));
case 'noiseletm'
q = randperm(n^2)';
Iml=realnoiseletm(eye(n^2)); %I matrix for PINV
Iml=Iml(q(1:m),:);
Iml=(Iml+1)/2;
case 'binarym'
Iml=double(rand(m,n^2)>0.5);
case 'realm'
Iml=rand(m,n^2);
case 'dctsm' %change m to closest square, use upper left square for DCT
nmv=(1:n).^2;
[~,nm]=min(abs(nmv-m));
m=nm^2;
Iml=zeros(m,n^2);
for k=1:m
v=zeros(nm^2,1);
v(k)=1;
vm=zeros(n,n);
vm(1:nm,1:nm)=reshape(v,[nm,nm]);
psi=dct2(vm);
Iml(k,:)=psi(:);
end
case 'dctzzm' %use jpeg zigzag DCT
Iml=dct_zigzag(n,m);
Iml=Iml-min(Iml(:));
end
if illum_type(end)=='m'
%I1=@(z) Iml*z; %ideal I
%It1=@(z) Iml'*z;
Im=scale_rows(Iml,scale)+vI*randn(m,n^2); %real I with scale and noise
I=@(z) Im*z;
%It=@(z) Im'*z;
Im2=Im+vI2*randn(m,n^2); %measured I for TGI
%I2=@(z) Im2*z;
%It2=@(z) Im2'*z;
else
I = @(z) I1(z);
%It = @(z) It1(z);
%I2 = @(z) I1(z);
%It2 = @(z) It1(z);
end

%Noise
% Im and I() - have scale and noise (scale and vI added)
% Iavev - is from measured I with measurement noise
% (vI2 added on top of scale and vI)
% Iavev1 - is from ideal I
% S, R, Save, Rave - all have noise added (vs on top of scale and vI)

T=img(:); %transmittance
S=I(T)+sqrt(vs)*randn(m,1); %bucket measurements
R=I(ones(n^2,1))+sqrt(vs)*randn(m,1); % reference bucket, sums each
illum. pattern (each row)
Rave=mean(R); %(scalar)
Iavev1=It1(ones(m,1))/m; %use ideal I
Iavev=It2(ones(m,1))/m; %measured I (TGI)
Save=mean(S); %average S (scalar)

```

```

tic
switch gi_type
case 'TGI'
O=(It2(S-Save)-Iavev*sum(S-Save))/m; %traditional GI (I-Iave)(S-Save)
case 'CGI'
O=(It1(S-Save)-Iavev1*sum(S-Save))/m; %computational GI
case 'DGI'
O=(It1(S-R*Save/Rave)-Iavev1*sum(S-R*Save/Rave))/m; %differential GI
case 'NGI'
O=(It1((S./R)-Save/Rave)-Iavev1*sum((S./R)-Save/Rave))/m;%normalized GI
case 'PINV'
O=pinv(Iml)*S; %pseudoinverse
case 'DPINV'
O=pinv(Iml)*(S-R*Save/Rave)/m; %differential pseudoinverse
case 'NPINV'
O=pinv(Iml)*((S./R)-Save/Rave)/m; %normalized pseudoinverse
case 'JPEG'
O=reshape(idct2(dct2(img).*ones_zigzag(n,m)),n^2,1);
%jpeg like compression, close to DCT illum
end
time=toc;
if do_denoise
O=denoise(reshape(O,n,n),img);
O=O(:);
end
O=imnormalize(O);
PSNR=-10*log10((sum((T-O).^2)/n^2));
end

function [ v2 ] = index( v,i )
v2=v(i,:);
end

function [ v2 ] = unindex( v,i,n)
v2=zeros(n,1);
v2(i)=v;
end

function [ y ] = imnormalize( x )
y=(x-min(min(x)))/max(max(x-min(min(x)))));
end

function [ m ] = scale_rows( m,s )
%scale_rows scales rows of m, starting from 1 to 1+s
[r,~]=size(m);
for i=1:r
m(i,:)=m(i,:)*((i*s/r)+1);
end
end

function y = realnoiseletm(x)
[m,n]=size(x);
y=zeros(m,n);
m=uint32(m);
for i=1:n
c = m - 1;
j=uint32(0);
mh=bitshift(m,-1);
while (j < mh)
k = bitxor(j,c);
y(j+1,i) = x(j+1,i) + x(k+1,i);
y(k+1,i) = x(j+1,i) - x(k+1,i);

```

```

    j=j+1;
end
d = bitshift(c,-1);
while (d > 0)
    j=uint32(0);
    while j < mh
        k = bitxor(bitxor(j,c),d);
        temp = y(j+1,i);
        y(j+1,i) = y(j+1,i) - y(k+1,i);
        y(k+1,i) = temp + y(k+1,i);
        j=j+1;
    end
    d = bitshift(d,-1);
end
end
end

function A=dct_zigzag(n,m)
d=1; %diagonal number
cnt=1; %counts measurements (rows of A)
up=1; %1 = col increasing, 0 = col decreasing
A=zeros(m,n^2);
r=d; %row index
c=1; %col index
am=zeros(n,n); %dct matrix
d_inc=1;
%figure(1) %debug
while cnt<=m;
    for i=1:d
        am(r,c)=1;
        %imshow(imresize(am,16,'nearest'),[]) %debug
        psi=dct2(am);
        am(r,c)=0; %init back to 0
        A(cnt,:)=psi(:); %vectorize dct and add to A
        cnt=cnt+1;
        if cnt > m %stop when have enough measurements
            break;
        end
        if up %update indices
            r=r-1;
            c=c+1;
        else
            r=r+1;
            c=c-1;
        end
    end
    if d==n %past half way
        d_inc=-1;
    end
    d=d+d_inc; %increment diagonal
    up=~up; %switch direction
    if up %init indices
        if d_inc==1
            r=d;
            c=1;
        else %past half way
            r=n;
            c=n-d+1;
        end
    else
        if d_inc==1
            c=d;
            r=1;
        end
    end
end
end

```

```

    else %past half way
        c=n;
        r=n-d+1;
    end
end
end
end

function am=ones_zigzag(n,m)
d=1; %diagonal number
cnt=1; %counts measurements (rows of A)
up=1; %1 = col increasing, 0 = col decreasing
%A=zeros(m,n^2);
r=d; %row index
c=1; %col index
am=zeros(n,n); %dct matrix
d_inc=1;
%figure(1) %debug
while cnt<=m;
for i=1:d
    am(r,c)=1;
    %imshow(imresize(am,16,'nearest'),[]) %debug
    %psi=dct2(am);
    %am(r,c)=0; %init back to 0
    %A(cnt,:)=psi(:); %vectorize dct and add to A
    cnt=cnt+1;
    if cnt > m %stop when have enough measurements
        break;
    end
    if up %update indices
        r=r-1;
        c=c+1;
    else
        r=r+1;
        c=c-1;
    end
end
if d==n %past half way
    d_inc=-1;
end
d=d+d_inc; %increment diagonal
up=~up; %switch direction
if up %init indices
    if d_inc==1
        r=d;
        c=1;
    else %past half way
        r=n;
        c=n-d+1;
    end
else
    if d_inc==1
        c=d;
        r=1;
    else %past half way
        c=n;
        r=n-d+1;
    end
end
end
end
end

function fTV0 = denoise( y, f0 )

```

```

niter = 800;
epsilon=0.01;
lambda=0.1;
tau = 2 / ( 1 + lambda * 8 / epsilon);
fTV = y;
err_best=-inf;
for i=1:niter
    Gr = grad(fTV);
    d = sqrt(sum(Gr.^2,3));
    G0 = -div( Gr ./ repmat( sqrt( epsilon^2 + d.^2 ) , [1 1 2]) );
    G = fTV-y+lambda*G0;
    fTV = fTV - tau*G;
    err = snr(f0,fTV);
    if err>err_best
        err_best=err;
        fTV0=fTV;
    end
end
end

function fx=grad(M)
fx = M([2:end end],:)-M;
fy = M(:, [2:end end])-M;
fx = cat(3,fx,fy);
end

function fd=div(Px)
Py = Px(:,:,2);
Px = Px(:,:,1);
fx = Px-Px([1 1:end-1],:);
fx(1,:) = Px(1,:);
fx(end,:) = -Px(end-1,:);
fy = Py-Py(:, [1 1:end-1]);
fy(:,1) = Py(:,1);
fy(:,end) = -Py(:,end-1);
fd = fx+fy;
end

function v=snr(x,y)
v = 20*log10(norm(x(:))/norm(x(:)-y(:)));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% GI_test.m: Reproduce Fig. 7 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear
close all

img=double(imread('cameraman.tif')); %img
n=32; %width of image
m=round(20*n^2); %number of measurements
gi_type={'CGI','TGI'};

% if vratio is low, that means vI2v (measured I for TGI) is high
% and other noise is low, means TGI will do bad
vs=0.5:-0.05:0.05; %std dev of noise added to bucket and reference bucket
vI=0.5:-0.05:0.05; %std dev of noise added to I
vI2=0.1:0.1:1; %std dev of noise added to measured I
nloops=length(vs);
ntypes=length(gi_type);
vratio=(vs+vI)./vI2;
[r,c]=size(img);

```

```

img = imresize(img,n/r);
PSNR=zeros(ntypes,nloops);
fprintf('Total = %d, ',nloops)
for loop=1:nloops
    rng('default');
    fprintf('%d ',loop);
    for i=1:ntypes
        [~,~,PSNR(i,loop)]=...
            GI(img,m,char(gi_type(i)), 'binarym',0,vs(loop),vI(loop),vI2(loop),0);
    end
end

figure
semilogx(vratio,PSNR,'LineWidth',1.5)
xlabel('\kappa')
ylabel('PSNR (dB)')
legend(gi_type,'Location','SouthEast')
fprintf('Done!\n',loop);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% GI test.m: Reproduce Figures. 15, 17, 22, 23, and 24 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear
close all

img=double(imread('cameraman.tif')); %img
n=32; %width of image
m=round((0.3:0.05:1)*n^2); %number of measurements
gi_type={'CGI','DGI','NGI','PINV'};
illum_type={'binarym','hadamardm','noiseletm','dctzzm'};
illum_title={'Binary','Hadamard','Noiselet','DCT'};
m_plot=round([0.3 0.7 0.9 1]*n^2); % measurements to plot

m_ploti=zeros(1,length(m_plot)); %create index of m_plot into m vector
for i=1:length(m_plot)
    m_ploti(i)=find(m==m_plot(i));
end
nillum=length(illum_type); %number of illum
nm=length(m); %number of measurements
ntypes=length(gi_type); %number of GI algorithms
[r,c]=size(img); %rows and cols of image
img = imresize(img,n/r); %resize image base on n
PSNR=zeros(ntypes,nm,nillum); %init result matrices
Om=zeros(n^2,nm,ntypes,nillum);
fprintf('Total = %d, ',nm*nillum)
timem=zeros(ntypes,nm,nillum);
for illumi=1:nillum
    for mi=1:nm
        rng('default'); %init random number gen
        fprintf('%d ',mi+(illumi-1)*nm) %progress
        for typei=1:ntypes
            %run GI sim
            [Om(:,mi,typei,illumi),timem(typei,mi,illumi),PSNR(typei,mi,illumi)]=...
                GI(img,m(mi),char(gi_type(typei)),char(illum_type(illumi)),0,0,0,0,0);
        end
    end
end

%% plot images based on m_plot list
for illumi=1:nillum
    figure
    ha = tight_subplot(length(m_plot),ntypes,[.01 .01],[.01 .05],[.1 .01]);

```



```

for mi=1:length(m_plot)
    for typei=1:ntypes
        axes(ha((mi-1)*ntypes+typei));
        imshow(reshape(Om(:,m_ploti(mi)),typei,illum_i),n,n,[],[])
        if mi==1 %only list gi_type for first row
            title([char(gi_type(typei)),sprintf('PSNR = %.1f
dB',PSNR(typei,m_ploti(mi),illum_i))])
        else
            title(sprintf('PSNR = %.1f dB',PSNR(typei,m_ploti(mi),illum_i)))
        end
        if typei==1 %only list sampling for first col
            ylabel(sprintf('%d%% Sampling',round(100*m(m_ploti(mi))/n^2)),...
                'FontSize',12,'FontWeight','bold')
        end
    end
end
set(gcf,'Position',[600 200 700 750])
end
fprintf('Done!\n')

%% Compare illum, organized by GI type
figure
subplot_list=[1 2; 3 4; 6 7; 8 9]; %each subplot spans 2 cols
line_list={'-','-.-','--',':'};
for typei=1:ntypes %GI type
    subplot(2,5,subplot_list(typei,:)) %legend will be on last subplot col
    for illum_i=1:nillum %illum type
        plot(100*m/n^2, PSNR(typei,:,illum_i),line_list{illum_i},'LineWidth',1.5)
        hold on
    end
    title(gi_type{typei})
    ylim([4 45])
    xlim([100*m(1)/n^2 100*m(end)/n^2])
    if mod(typei,2)==1
        ylabel('PSNR (dB)')
    end
    if typei>2
        xlabel('Sampling (Percent)')
    end
end
end
s=subplot(2,5,5);
s.Position=[0.8137 0.8548 0.031 0.03870];
for illum_i=1:nillum
    plot(1,1,line_list{illum_i},'LineWidth',1.5)
    hold on
end
l=legend(illum_title,'Location','North');
l.Position=[0.7590 0.7565 0.2014 0.1698];

%% Compare types, organized by illum
figure
subplot_list=[1 2; 3 4; 6 7; 8 9]; %each subplot spans 2 cols
for illum_i=1:nillum %illum type
    subplot(2,5,subplot_list(illum_i,:)) %legend will be on last subplot col
    for typei=1:ntypes %GI type
        plot(100*m/n^2, PSNR(typei,:,illum_i),line_list{typei},'LineWidth',1.5)
        hold on
    end
    title(illum_title{illum_i})
    ylim([4 45])
    xlim([100*m(1)/n^2 100*m(end)/n^2])
    if mod(illum_i,2)==1
        ylabel('PSNR (dB)')
    end
end

```

```

end
if illumi>2
    xlabel('Sampling (Percent)')
end
end
s=subplot(2,5,5);
s.Position=[0.8137 0.8548 0.031 0.03870];
for typei=1:ntypes
    plot(1,1,line_list{typei},'LineWidth',1.5)
    hold on
end
l=legend(gi_type,'Location','North');
l.Position=[0.7590 0.7565 0.2014 0.1698];

```

## List of Symbols, Abbreviations, and Acronyms

---

2-D	2-dimensional
3-D	3-dimensional
CS	compressive sensing
CGI	computational ghost imaging
CPU	central processing unit
DCT	discrete cosine transform
DGI	differential ghost imaging
DPINV	differential pseudo-inverse
FPA	focal point array
GI	ghost imaging
JPEG	Joint Photographic Experts Group
NGI	normalized ghost imaging
NPINV	normalized pseudo-inverse
PC	personal computer
PINV	pseudo-inverse
PSNR	peak-signal-to-noise ratio
RAM	random access memory
SLM	spatial light modulator
TGI	traditional ghost imaging
TV	total variation
WHT	Walsh–Hadamard Transform

1 DEFENSE TECHNICAL  
(PDF) INFORMATION CTR  
DTIC OCA

1 CCDC ARL  
(PDF) FCDD RLD CL  
TECH LIB

23 CCDC ARL  
(PDF) FCDD RLW LF  
B ALLIK  
B J ACKER  
T G BROWN  
S BUGGS  
E BUKOWSKI  
J COLLINS  
J CONDON  
B DAVIS  
M DON  
D EVERSON  
R HALL  
J HALLAMEYER  
M HAMAOU  
T HARKINS  
M ILG  
B KLINE  
J MALEY  
C MILLER  
B NELSON  
D PETRICK  
K PUGH  
N SCHOMER  
B TOPPER