Embedding Meta Information into Visualizations

Alok Hota, *Student Member, IEEE* and Jian Huang, *Senior Member, IEEE*

Abstract—In this work, we study how to co-locate meta information with visualizations by directly embedding information into visualizations. This allows for visualizations to carry provenance and authorship information themselves for reproducibility. We call these self-describing visualizations–reproducible, authenticatable, and documentable. Self-describing visualizations can be used to extend existing visualization provenance systems. Herein, we start with a survey of existing digital image watermarking literature. We search for and classify watermarking algorithms that can support scientific visualizations. Using our payload-resilience testing framework, we evaluate and recommend algorithms supporting various use cases in the payload-resiliency space, and present guidelines for optimizing visualizations to improve payload capacities and embedding robustness. We demonstrate the efficacy of self-describing visualizations with two sample application implementations: (1) adding an embedding filter as a part the standard rendering pipeline, (2) creating a web reader to automatically and reliably extract provenance information from scientific publications for review and dissemination.

Index Terms—Scientific visualization, reproducibility, visualization systems, digital image watermarking

1 Introduction

Advancement of science depends on having results that can be reproduced, peer reviewed, and improved. As visualizations are an important part of today's scientific results, they need to meet the same veracity standards required of all other scientific results.

A key step in doing so is to capture and record the provenance information describing how each visualization has been created. Because the process to create visualizations can be complex, the provenance information is often neither compact nor easy-to-manage. Thereby we often run into scenarios where the visualizations are dissociated from the corresponding provenance information.

Furthermore, scientists and researchers often revisit previous results days or months later. Without meticulous records-keeping, information on how a visualization was created, its purpose, and its story can be lost to time easily. Published visualizations, such as those in papers and presentations, suffer from this problem, too, as there is insufficient page space to fully describe visualization provenance. As these papers and presentations are shared, knowledge of how a particular visualization was created is lost. Further, one will need to undergo a cumbersome and manual process to recreate results from a publication, typically.

In this work, we explore how to embed meta information directly into visualizations, i.e., the images themselves. We call these self-describing visualizations–visualizations that are reproducible, authenticatable, and documentable. This allows provenance information, author information, authenticity information, etc. to be co-located with the visualizations they describe. The co-location in self-describing visualizations additionally prevents unintentional, or even intentional, changes to the provenance information.

Our goals are to: (1) embed information with minimal visual impact on the visualization, (2) embed information such that it travels with the visualization through publications, presentations, emails, and other media, (3) create a process that is feasible as part of the visualization pipeline (i.e., not as a manual process), and (4) extract the embedded information reliably and enable new applications.

Our method of achieving these goals is to use digital image watermarking. Watermarking is a well-studied concept in image processing, but so far very rarely used for visualizations. Through our survey of the watermarking literature, we discovered that most existing watermarking methods only qualitatively embed information in images. Embedding provenance information into visualizations requires quantitative information embedding–embedding with bit-for-bit accuracy upon extracting. Another discovery is that watermarking literature tends to focus on photographs of natural subjects with a lot of detail and complexities, which makes hiding information easier. Visualizations have very different characteristics from photographs.

To understand characteristics of the data to be embedded, we examined various existing ways of collecting provenance information, ranging from lookmarks in ParaView [1], session files in VisIt, VT files from VisTrails [2], and linked documentation from F1000Research, a recent online publication platform. We discovered that provenance information varies from tens of bytes to several kilobytes. Visualization watermarking may need to provide different design choices depending on the amount of information to embed. As a result of this work, the algorithms we suggest are Entropy Thresholding and Least Significant Bit with Optimal Pixel Adjustment Process to cover these use cases.

• *The authors are with the Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN 37996. E-mail: ahota@vols.utk.edu, huangj@utk.edu.*

Manuscript received 29 Oct. 2018; revised 6 Apr. 2019; accepted 20 Apr. 2019. Date of publication 14 May 2019; date of current version 6 Oct. 2020. (Corresponding author: Alok Hota.)
Recommended for acceptance by E. Gobbetti.
Digital Object Identifier no. 10.1109/TVCG.2019.2916098

We include a brief survey of existing image-processing watermarking literature in Section 3. In Section 4, we classify visualization use cases within the two-dimensional payload-resilience space. For each class, we test, analyze, and recommend the most viable algorithms in Section 5. In Section 6, we describe and demonstrate two reference implementations of self-describing visualizations. First, a rendering engine that includes watermarking as a natural final step of rendering. Second, a web service that automatically extracts watermarked information from PDF files that use self-describing visualizations as figures. Finally, in Section 7, we discuss our findings using external supporting techniques, such as error-correcting codes, and possibilities for metadata descriptors.

2 Background

Having reproducible results is a cornerstone of science. This applies to visualizations, where all techniques rely on information external to the visualization. If a visualization is copied away from its original source, all ties to the information are lost. Trial-and-error attempts to recreate such a visualization can require parallel computation on a supercomputer [3]. self-describing visualizations–those which are reproducible, authenticatable, and documentable–are to extend existing provenance technologies by co-locating visualizations with their provenance information. Thus embedding a verifiable ground truth and historical documentation into the visualization itself.

2.1 Reproducible Research

An integral part of the scientific procedure is being able to reproduce results from previous works. The purpose is not to show mistrust for the scientists or results, but to assert the veracity of the work. Reproducibility, or full replicability, is an important facet of robust research in all scientific fields. However it is often cited as lacking in publications [4], [5], [6], and studies assessing reproducibility are difficult to publish in top journals [7].

To address this gap, the National Science Foundation (NSF) supports projects aiming to improve reproducibility in computer systems and networking research [8]. New calls require thorough data management and open access plans to improve reproducibility studies. Additionally, there are open solicitations by the NSF for Big Data projects focusing on reproducibility and replicability in data science [9]. Peng describes the need for a "culture of reproducibility" in the computational sciences, and introduces a reproducibility spectrum [6]. In this spectrum, publications should come with executable code and data to meet the "gold standard", which would allow for results to be reproduced.

2.2 Reproducibility and Provenance in Visualization

Visualizations enable communicating complex features in scientific data and as such should be held to the same standard as science. Perkel discussed interactive information visualizations in online publications supported by Plotly and F1000Research [10] that improve reproducibility. For scientific visualization, ParaView supports the use of *lookmarks*, which can replicate full application state to return to a saved point in analysis [1]. The analog in the VisIt visualization package is a session file, which can be created at any time to provide a snapshot of the current analysis performed. This can be used as a way of sharing how a final visualization was created if provided with the data and a running application to meet the gold standard.

More broadly, a way to maintain reproducibility in visualization research is provenance, which generalizes capturing histories of analyses. Ragan et al. characterize the various types and purposes of provenance [11]. A widely-known system in visualization for reproducibility is VisTrails [12], which allows visualization reproduction [2]. In particular, VisTrails captures actions during the exploration and analysis process to create a workflow describing how a visualization was achieved. This allows for maintaining the authenticity of a visualization (i.e., who created this visualization?) as well as documenting a visualization (how was it created?) in a "vistrail" object. Vistrail metadata can be included in a PDF file, allowing the final document to be linked to the visualization source via a LATEX package.

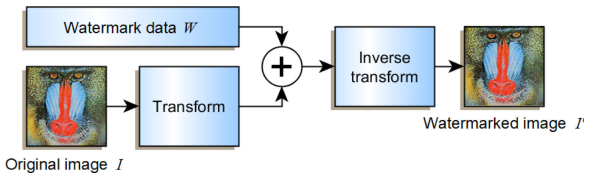
Of course, both the provenance information and the original computing environment are required for reproducibility. An extension to VisTrails creates executable papers, wherein full executable environments are available within a PDF [13]. The full environment allows for simplified linkages between VisTrails and other computational platforms, such as ALPS [14]. To this end, vistrail objects are mainly meant for computer codes to understand and automate a toolchain to recreate visualizations. In contrast, self-describing visualizations can contain general, human-readable data, that are permanently co-located with each visualization. These two techniques should be used in conjunction.

2.3 Image File Metadata

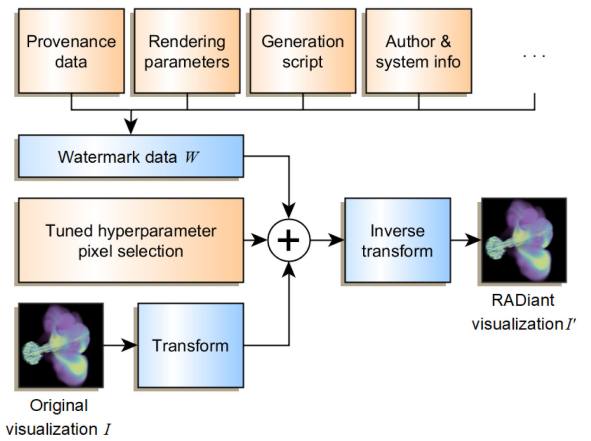
In a way, we can consider the generalization of visualizations as images. A simple method of including any information with any image is to add file metadata. The metadata could contain authorship information, rendering parameters, analysis, etc. and would be directly tied to the file. This is the use case for Exif metadata in digital photography [15]. In a LATEX-based publication, this metadata would be preserved, allowing any reader to extract it.

The Exif standard extends JPEG, TIFF, and WAV. Of these only JPEG is a useful candidate in visualization. For broader use in visualizations, the PNG standard would also have to be extended to include the metadata. The XMP standard fits this need as it allows for more media filetypes, and "sidecar" metadata for unsupported types [16], [17].

There are fundamental issues with file metadata approaches. If the file is copied or converted to another file type, the metadata may be lost. Metadata may be purposefully stripped from files by certain applications at the user's discretion. Using an image-space watermarking approach generalizes the survivability of the embedded information. While image file metadata would survive embedding in a PDF, it would not survive indirect transmissions, such as screenshots. In this work, we modify existing bitmap, PNG, or JPEG images on a pixel-wise basis. Stripping the embedded data is not possible without altering the image itself, which defeats the purpose of the visualization. Data from these images can be read even after a copy since information is embedded within pixel values or frequency information.



(a) Possible transformations include discrete Fourier, cosine, and wavelet transforms. Some algorithms do not use a transformation and so this step in the workflow is a no-op.



(b) The watermark data is directly related to the image unlike many traditional use cases. Additionally, for methods with a transformation step, we need to tune hyperparameters to select good regions of the image.

Fig. 1. (a) is an overview of the watermark embedding process, and (b) shows the changes needed to target visualizations. We focus on symmetric methods in this work, thus extraction is the reverse of embedding.

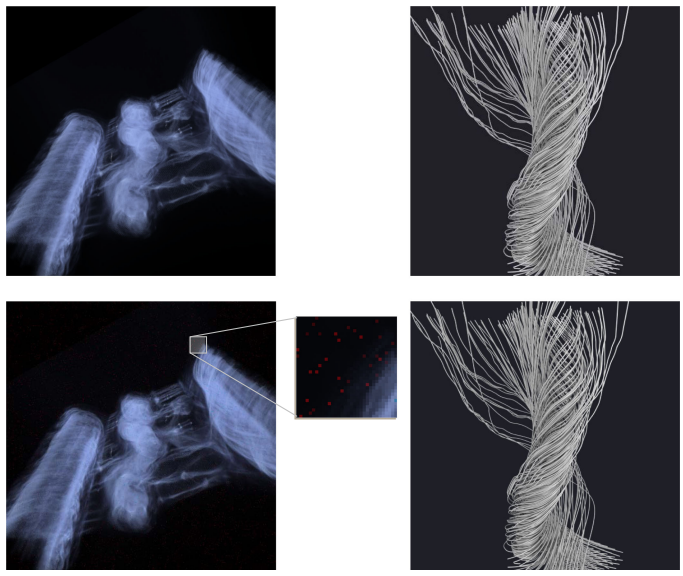


Fig. 2. Examples of original visualizations (top) with their watermarked counterpart (bottom). (a) contains a qualitatively perceptible watermark (red noise pattern), which can occur when the method's hyperparameters are set too high. (b) contains a qualitatively imperceptible watermark. Quantitatively, the image has a high peak signal-to-noise ratio of 43.15 when compared to the original, meaning the watermarked image is visually similar to the original.

3 Survey of Digital Image Watermarking

Digital image watermarking (DIW) is a type of *information hiding* within images. In the image processing community, DIW is well studied, with a variety of techniques. Herein we start with an overview of DIW as a whole and narrow down to techniques that are viable for visualization.

Hiding information in an image is known as *embedding* a watermark. A generalized conceptual process for embedding is shown in Fig. 1a. Techniques may differ in how they transform and inverse transform the image. Some techniques do not transform the image at all, so these steps would be no-operations.

Retrieving hidden information from an image is known as *extracting* a watermark. In general, extraction is the reverse of embedding. That is, these techniques are generally symmetric. Watermarking techniques based on cryptographic methods may use asymmetric techniques [18], [19]. Such methods are attractive, but would require public keys external to the visualization to decrypt embedded information, which can be lost when sharing the visualization.

Watermarking for visualizations takes some special consideration. We identified subprocesses of the embedding and extracting process in order to create self-describing visualizations. Fig. 1b shows the crucial components in context of the original watermarking workflow. The meta information blocks (provenance data, rendering parameters, etc.) provide the information needed to reproduce, authenticate, and document a visualization. The specific data can be sourced from the visualization application itself if watermarking is performed as a post-processing filter during rendering. Not all types of meta information shown in Fig. 1b must be used. Therefore the amount of embedded meta information depends on the specific use case. Selecting the pixels to embed data within is also crucial, and varies image to image. We discuss these components in more detail in Section 4.1.

3.1 Imperceptibility

Digital image watermarking methods aim to *imperceptibly* hide data within regions of an image. Qualitatively, perceptibility can be measured by simply looking at watermarked images, for example. In Fig. 2a the visible red noise makes it clear that the image has been modified, whereas Fig. 2b looks like an untouched image (though it does contain a watermark). Quantitatively, perceptibility can be measured by calculating the peak signal-to-noise ratio (PSNR) compared to the original image. This is often used to benchmark the performance of a watermarking technique. Fig. 2b has a fairly high PSNR of 43.15, meaning it is close to the original compared to the lower 27.61 for Fig. 2a.

Imperceptibility can be achieved by exploiting the human visual system (HVS) in various ways. For example, data can be encoded within least significant bits where the HVS is not sensitive enough to detect variations in color [20], or to leverage the HVS' low sensitivity to alterations in high-frequency luminance regions as compared to low-frequency luminance regions [21]. In fact, the JPEG compression standard targets similar regions of an image since loss of information in those regions is less noticeable [22]. Fig. 3 shows a comparison between an original visualization of supernova data compared to an imperceptibly watermarked version using a watermarking technique with a JPEG-like method to select data regions.

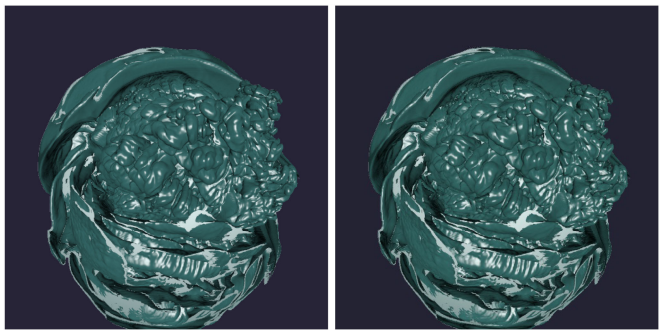


Fig. 3. Left, an original visualization of a supernova simulation isosurface. Right, the same visualization watermarked with the ET [46] method, which uses the 2D DCT space for embedding data. This allows it to resist JPEG compression, while remaining imperceptible by taking advantage of the human visual system's low sensitivity to alterations in high frequency regions.

3.2 Extractor versus Detector

During extraction, a watermark may be *extracted* or *detected*. Detection simply requires that a watermark be detected in the image at all, and does not require any information to be retrieved. This is essentially 1-bit watermarking, as the detector returns a boolean value. Techniques may be extended to multiple bits [23]. Even though the embedded bit(s) may be highly resilient [24], detection-only methods are not applicable to our use case. For this work we focus on multi-byte methods with data extraction because we are targeting scripts or keys as the embedded data.

Changes to a watermarked image, whether accidental or intentional, are called *attacks*. Attacks to an image may result in *insertions* or *deletions* to watermark data at the extractor. An insertion occurs when the extractor determines that a bit (i.e., a 0 or 1) exists in the image when the watermark did not actually contain this bit. In contrast, a deletion occurs when the extractor skips a bit that did exist in the watermark.

A critical requirement of DIW methods is *synchronization* between the embedder and extractor. Synchronization means that the embedder and extractor hide and seek data in the same locations. With synchronization, the increase in error (insertions) or loss of information (deletions) may be recoverable through error coding or even ignored. However, desynchronization is catastrophic to any method.

3.3 Variety of Resilience Profiles

The *resiliency* of a watermarking method measures resistance to various attacks. Attacks could include compression, introduced noise through transmission, etc. *Fragile* algorithms are not resilient, a quality sometimes desired if checking for (un)authorized usage [25].

A watermarking technique's resiliency depends on a number of hyperparameters, e.g., which color channels or bit planes are modified, which regions are selected, etc [26]. Techniques that transform the image may acquire resiliency to some attack profiles through properties of the transformation space. Common benchmarking attacks include compression, noise, affine transformations, cropping, and color alteration [27]. A technique is considered resilient if an embedded watermark can be reliably extracted from an image in the presence of an attack.

Many methods, especially those that embed images into images, have a relaxed stance on error in the watermark [28], [29]. In these algorithms, some degree of error is acceptable because it is implicitly understood that a human will be in the loop to assess the extracted watermark. We have a high bar for resiliency when used for visualizations. Because we are embedding scripts or authorization keys, we require *bit-for-bit* accuracy at the extractor.

3.4 Image Space versus Transformation Space

Simple DIW methods operate directly in image space and tend to have high payload size, but are more fragile. Some minor alteration to the pixel data, such as compression, can destroy the embedded data.

Advanced DIW methods use a transformation space to embed watermark information. Common transformation spaces include discrete Fourier, wavelet, and cosine spaces. Each transformation space comes with a set of pros and cons as well as a certain set of attacks to which it is resilient. Some methods may use hybrid spaces to acquire even better resilience [30], [31], [32].

3.4.1 Fourier Space Embedding

Discrete Fourier transform (DFT) space is a popular choice for data embedding [24], [33], [34], [35], [36], [37]. Watermark bits can be embedded into selected frequency coefficients directly in DFT space. This avoids alterations caused by compression algorithms, which primarily target high frequencies, and avoids perceptual changes to low frequency regions of the image.

DFT embedding is particularly attractive as it allows for resiliency to *geometric* attacks-rotation, scale, and translation (RST) through properties of the Fourier transform [33]. RST resiliency often allows images to be physically printed and still retain extractable watermarks [38]. Furthermore, since the transformation represents the data completely in frequency space, embedded bits may be spatially distributed to different regions of the images.

For visualizations, we do not consider RST attacks to be typical. This is especially the case since many publications today that would include visualizations are published electronically as PDFs. As long as the PDF is not compressed, the original image can be parsed from the PDF, which eliminates any scaling issues. As a shortcoming, however, robust Fourier methods could introduce noticeable artifacts depending on the mapping used [33].

3.4.2 Wavelet Space Embedding

2D discrete wavelet transform (DWT) may be used for embedding as well [31], [32], [39], [40], [41], [42]. The DWT performs a horizontal and vertical downsampling operation on the image, creating four subregions. This process can be repeated on the lowpass subregion and produce multiresolution wavelet decomposition [39], [41]. Watermark data can be embedded in the low frequency information and the image is reconstructed. The wavelet used for downsampling could be a simple Haar wavelet, or a Daubechies-family wavelet as in the JPEG 2000 compression algorithm [43].

Embedding in the DWT space alone will result in data being spatially spread through the reconstructed image. If data is embedded in the kth resolution decomposition, then that data will affect a 2^k area in the image. This may lead to

highly noticeable artifacts. Moreover embedding in the wavelet transformation space may help with JPEG 2000 compression resiliency, however this is not the typical compression algorithm used to save JPEG images.

3.4.3 Cosine Space Embedding

The discrete cosine transform (DCT) is a Fourier-based method which decomposes a signal into sinusoids with only real components. 2D DCT is used by the JPEG standard [22] to compress images, as well as the MPEG standard for video [44]. This is an advantage for watermarking methods using this transform [28], [45], [46], [47], [48]. Watermark data can be embedded in the cosine space such that the JPEG compression process has minimal effect to the data. Fig. 3 shows an example of a visualization with data embedded in the DCT space which can resist compression.

In watermarking, this transform is generally performed upon small blocks of the image, usually 8×8 blocks [28], [49]. Data can be embedded in one or more of the resulting 64 coefficients within these blocks. It is not necessary for every block to be used for embedding. Instead, many techniques choose certain blocks that contain either smooth [48] or textured [49] regions. While this reduces the possible locations for data to be hidden, it increases the robustness of the watermark.

Since the blocks are spatial subdivisions of the original image, the watermark data is embedded in certain spatial regions of the image. This can lead to weakness to cropping, but we do not believe cropping is a likely attack in visualizations. We also do not believe heavy compression is a likely attack, as this would destroy useful detail in the visualization. However since saving visualizations as JPEG is common, resilience to light compression with default JPEG encoding parameters is useful. Thus we take DCT-based methods into consideration.

Most existing watermarking methods using DCT space target grayscale images. For full-color visualizations, we use the luminance channel for embedding. Individual color channels may or may not contain enough information depending on color map, shadow, background color, etc. to guarantee that one channel will work for all visualizations.

3.5 Variety of Embedding Needs

The actual data that will be embedded is of special interest. *Payload size* refers to the maximum number of watermark data bits that can be embedded within an image. We require methods that can effectively hide tens of bytes to multiple kilobytes of payload to cover various use cases of visualization meta information.

Using the nomenclature from Cox et al. [25], there are four classifications of information hiding based on the message itself. The first two are watermarking and non-watermarking systems. These are defined as systems where the data is related to the cover image and systems where the data is unrelated to the cover image, respectively. The next two are steganographic and non-steganographic systems. These are defined as systems where the message's existence is secret and systems where the message's existence does not need to be secret.

For watermarking visualizations, we need a non-steganographic watermarking system. The information is related to the cover image (provenance data) and it does not need to be a secret that the watermark is present (though it should be imperceptible).

3.5.1 Common Watermark Data Types

Many techniques focus on image-in-image watermarking [28], [36], [50], [51]. These systems may be copyright-related, where the watermark image is a logo of the authoring party. Most methods use binary images, commonly for signatures or simple logos, however some methods can embed full-color watermark images as well [52], [53]. We can generalize any watermark data type as an arbitrary bit sequence. In theory this makes any image-in-image technique as viable as other general methods.

However, there is a significant drawback with image-in-image watermarking techniques. There is an implicit understanding that the extracted watermark is only qualitatively assessed. That is, it is deemed acceptable that an extracted image contains some errors, insertions, or deletions because the human in the loop can still judge its visual similarity to the intended watermark [29].

Since we are hiding provenance information, we require bit-for-bit accuracy. Methods could use fingerprints [28] to check if the image has a watermark or if it is altered. But ultimately the method should have enough resiliency for the bit sequence to arrive unaltered.

3.5.2 Quantity of Embedded Data

The maximum payload size of a cover image can depend on a few factors. The first and most obvious is the size of the cover image. For a $w \times h$ image with c channels, the maximum payload size for an image-space method could be calculated as $w \times h \times c$ assuming 1 bit is embedded per pixel per channel.

Transform-space methods may theoretically have the same maximum payload size, but they are often used to embed far fewer bits with a higher resiliency. Many transform-space methods are *image-adaptive*; they restrict possible locations to hide data based on the contents of the image.

If the use case in visualization is to store a full script or program, an image-space method will be required. If a higher resiliency is needed, it is likely that a transformation-space method will be required at the cost of payload size.

3.5.3 Watermark Data Transformation

The watermark data may be transformed before embedding. We briefly discuss two general classes of watermark data transformation: those that transform the data signal and error-correcting codes.

A common signal transformation is *spread-spectrum* coding [54]. This paradigm models the watermark data as a signal and spreads it over a wider frequency space [55]. One method of spreading the watermark data is to multiply the incoming watermark bit stream by a pseudorandom number stream in the embedder. The extractor then needs the seed to regenerate the same pseudorandom number stream to maintain synchronization. Spread spectrum coding lends itself well to frequency modulation with the image, i.e., embedding within the Fourier domain. However, spread spectrum methods often rely on knowing the original image

and/or watermark so that the noise can be removed through correlation [54].

Another simple method of transforming the watermark data is using error-correction codes. These are useful in correcting bitwise errors in the extracted watermark data. A common method is Reed-Solomon (RS) coding [56]. In short, RS coding in the embedder takes three parameters: s, the symbol length in bits; k, the message length in number of symbols; and n, the length of coded message in symbols. This would support correcting up to $\frac{n-k}{2}$ errors. The extractor needs s, k, and n to maintain synchronization.

Error coding may be useful for transformation-space methods to acquire extra resiliency. However, if the extractor is expecting an n-bit encoded message, a number of insertions and deletions can destroy the error correcting capabilities as the full message may not be read. Theoretic frameworks exist for correcting multiple variable-length insertions and deletions [57], [58], [59]. In this work, we directly embed unaltered binary information into visualizations to test the applicability and feasibility of watermarking visualizations.

3.6 Variety of Extractor Needs

Three classes of watermarking methods exist: *non-blind*, *semi-blind*, and *blind* [27], [60]. These refer to how much information is required by the extractor. Pieces of information include I, the original cover image; W, the original watermark data; I', the *stego-image*, the image with an embedded watermark; and W', the extracted watermark data.

3.6.1 Non-Blind and Semi-Blind Watermarking

Non-blind methods expect the most amount of information during extraction. Primarily, they require I, from which the extractor can subtract the cover image from the stego-image, revealing W'[54]. Semi-blind methods require W during extraction, but do not need I [29]. This is generally the case where the watermark is the same, regardless of the image. Non-blind and semi-blind methods are not viable for self-describing visualizations, because visualizations could be from any source, author, or publication. The data embedded within each visualization is specific to that visualization.

3.6.2 Blind Watermarking

Fully blind methods do not have access to either I or W during extraction [25]. Thus the extractor has no notion of what the original image was or what the watermark could be. Some known scheme must be used in the embedder to choose locations, whether by random number generator or by image masking. This scheme is key between embedder and extractor synchronization.

Blind methods are prone to desynchronization as the extractor can rely only on searching in the correct locations to find embedded data. Error correction codes may be used to mitigate errors in the bit stream, but can still be thwarted by insertions and deletions. Thus, blind watermarking can be lower resiliency than non-blind methods.

An interesting side effect of blind extraction is that some data will always be returned, because there is no longer any notion of validity of the data being extracted. Without error coding, fingerprinting, or some other validation, the extractor must assume anything it processes is watermarked.

Blind methods are the most appropriate for visualization use, since they allow for the original source and author to be decoupled from the visualization end product. That is, there is no need to maintain a database of every original image and/or every known script, executable, etc.

4 Self-Describing Visualization Requirements

In this section we describe the requirements for watermarking visualizations based upon the literature survey in Section 3. We discuss the considerations for three important facets of creating self-describing visualizations. These are heavily dictated by the use case the visualization expert or scientist is targeting.

4.1 Requirements for Watermarking Visualizations

With the wealth of research in DIW from the image processing community, there is a large number of hyperparameters to tune. Visualizations are very different from the typical photographs in watermarking literature. Here, we list requirements and assumptions for using watermarking algorithms with visualizations, and choose four methods that fit these requirements to demonstrate self-describing visualizations.

The first requirement is a fully blind watermarking algorithm. This is because access to neither I nor W can be guaranteed. Suppose a figure in a journal publication is watermarked. From the reader's perspective, the original image may exist elsewhere, but the paper only contains the altered image.

Second, we consider that for visualization, the most important resilience is to compression. Cropping attacks are less important because in many cases the full image is available. For example, in a publication prepared with LATEX, the full image is stored in binary form within the PDF. We consider geometric attacks less relevant, since visualizations are created and stay within the digital domain.

Third, we require a method that can embed and extract numerous bytes with bit-for-bit accuracy. This requires that the embedder can successfully find multiple locations for data hiding within the cover image and that the extractor can correctly extract from these same locations. Insertions and deletions due to desynchronization will cause catastrophic errors. We leave the option open for having flexible payload size, which can range from authentication tokens to full scripts. This requires the image to contain enough viable locations for data. This is not a concern for image-space methods, but certain transformation spaces make this difficult depending on the visualization.

Finally, we require a method that can be applied to color images. Many works in the literature demonstrate and evaluate a method using only grayscale images. Multiple channels do not present much issue with image-space methods as there is a relaxed selection process. However methods that use a transform space are already performing a floating point operation with bit manipulation on selected pixels. Using a colorspace conversion is a viable option, but the watermark should survive this extra conversion process.

4.2 Data for Embedding

The data to embed should contain information needed by an external user to recreate the visualization. There is no

inherent restriction on what type or format of data is used for embedding. This notion is abstracted away by the fact that the embedder is ultimately taking a bit stream W and modifying pixels or regions of the image with these bits.

We can directly embed a script used to generate a visualization into the visualization itself. This could be a Python script used for VisIt or ParaView, an executable VTK C++ program, a Tapestry [61] configuration file, etc. To accommodate a script, a watermarking method that can embed $\geq$4 KB is preferred. If a level of secrecy or authentication is desired, the watermark data can be an auth token, an access-controlled database key, or similar. In this case, the chosen method need only embed 16 bytes, for example.

The amount of authorship or documentation information portrayed can be configured as well. An included script may contain the author's name, a link to where the data can be accessed, and possibly any publications for citations. This is ideal to match the reproducibility golden standard.

4.3 Types of Visualizations

Watermarking algorithms that use a transformation space often have a method for selecting "good" regions in an image for embedding data. The type of visualization has a significant impact on this selection process.

For example, the Entropy Threshold method calculates the entropy, or amount of information, within blocks in the cosine transform space [38]. After dividing I into 8×8 blocks, each block B has 2D DCT applied to it. The entropy E for a block B is

$$E_B = \sum {B_{ij}}^2, \quad \forall\, i, j \in 1, 2, \ldots, 7. \tag{1}$$

B_{00} is not considered in this calculation as it is the DC coefficient. Given a chosen threshold T, if $E_B \geq T$ before *and* after embedding data, B is considered a good block. This translates to each block in the image requiring enough high frequency luminance information to surpass T.

Entropy is quite different between natural photographs, volume rendered data, and surface rendered data. For example, Fig. 4a shows the "Mandrill" image, a standard test image in the image processing community. It is very common to see this image used as a benchmark for a watermarking algorithm. It contains high frequency luminance information spread spatially throughout the image, which makes it a good candidate for transform-space methods.

In contrast, a volume rendered visualization of a CSAFE heptane gas simulation is shown in Fig. 4b. There are very few regions containing high frequency information. The color map used for a visualization plays a significant role here. Research shows that perceptually linear luminance in a color map is preferable to avoid any nonexistent features from being created by the color map itself [62], [63]. This means that apart from edges where a volume render is composited with background color, there is likely *no* region with high luminance contrast.

Using the formula in Equation (1), the Mandrill has an average entropy across all 8×8 blocks of $E_M = 44223.18$, with a minimum of 390.94 and maximum of 243657.36 In comparison, the heptane volume render has an average entropy of just $E_{HV} = 3480.09$, with a minimum of 0 and maximum of 272796.23.

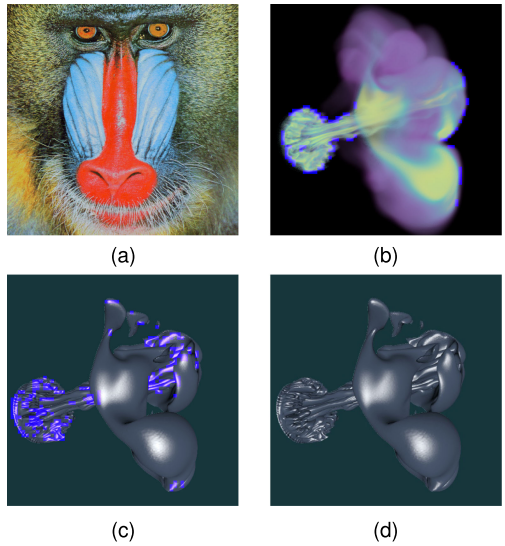


Fig. 4. (a) the "Mandrill" image, a common test image for watermarking methods. There is high frequency information throughout, making it ideal for methods like ET to hide data. (b) is a volume rendered visualization of the CSAFE heptane gas simulation. There are very few regions of high luminance frequency. Blocks containing a higher entropy than the average Mandrill entropy are highlighted in blue. (c) and (d) are the same heptane data rendered as an isosurface. Note that the surface has many more high frequency luminance blocks, allowing more data to be hidden.

Although the heptane's maximum is higher than the Mandrill, overall entropy is skewed towards the minimum. This is caused by the background, which as a flat color contains only a DC frequency component, or 0 entropy. To show the skew, Fig. 4b shows the heptane visualization with block entropies $> E_M$ highlighted in blue. Note that there are only a few blocks, all located around the edge.

Fig. 4c shows a surface render of the heptane data with an isovalue of 64, with blocks that have entropies $> E_M$ highlighted. The average entropy of this image is $E_{HS} = 9415.08$, a minimum of 0 again, but a much higher maximum of 378161.44. With specular lighting to boost high frequency regions, there are more blocks that stand on par with the Mandrill, but still not many blocks for embedding large amounts of data. This makes pixel selection in transform space methods much more difficult, leading to hyperparameter tuning.

4.4 Available Choices for Watermarking Algorithm

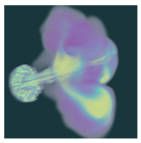
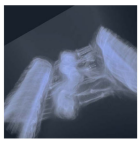
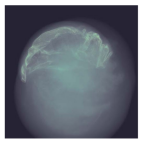
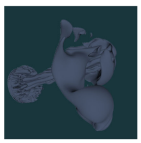
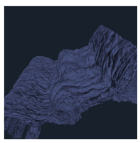
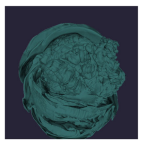
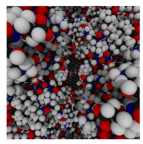
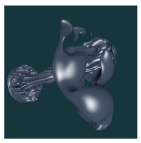
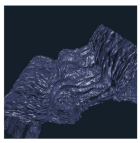
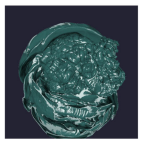
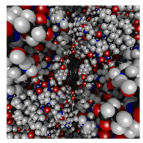
When choosing a watermarking algorithm, the primary trade-off is between payload size and the level of resilience. Therefore, if a 64 KB script is to be embedded, resiliency will be likely diminished. Similarly, strong resiliency will mean that only a few bytes can be embedded.

For visualizations, the most important resiliency profile is resistance to compression. On one hand, if it is known that a visualization will or could be compressed, it is more feasible to be embed a token with a transform-space method. On the other hand, if it is known that a visualization will not be compressed and would remain as a bitmap, then image space algorithms with larger payload size could be used to store full scripts or configurations.

5 Visualization Watermarking Guidelines

We started with an initial pool of 34 methods [18], [20], [24], [28], [29], [30], [31], [32], [33], [34], [35], [36], [37], [39], [40],

TABLE 1
Sample Visualizations Selected from Our 86-Image Suite Used to Test the Chosen Algorithms

	CSAFE heptane gas combustion	magnetic reconnection	SciDAC supernova	CroV giant virus	CM1 tornado sim
volumetric features low opacity					
surface features $k_s = 0$					
surface features $k_s = 0.4$					

These images represent various types of visualizations with different features. The ET method is successful primarily with surfaces with $k_s = 0.4$, which provides a good range of frequencies for information hiding. The LSB and HS methods worked on all visualizations, as they have non-image-adaptive pixel selection processes in image space. The PIC method was unsuccessful in extracting from any of the visualizations, as it suffered from insertions and deletions at the encoder.

[41], [42], [45], [46], [47], [48], [50], [51], [52], [53], [54], [64], [65], [66], [67], [68], [69], [70], [71] from our survey of the literature. We narrowed the pool down based on criteria from Section 4, eliminating non- and semi-blind methods, methods not resilient to compression, methods without bit-for-bit accuracy at the extractor, methods not extensible to color images, methods which we were not able to reproduce based on the description, and methods which either left noticeable artifacts or had low payload size.

We were left with four methods useful for testing self-describing visualizations: two image-space methods (LSB and HS) and two transform-space methods (ET and PIC). We have two main use cases: larger payload size for scripts and smaller payload size for tokens. We tested with a number of visualizations using different datasets, rendering styles, and rendering parameters.

The two image-space methods are Chan and Cheng's Least Significant Bit Substitution with Optimal Pixel Adjustment Process [70] (LSB) and Ni et al. Histogram Shifting [67] (HS). These were chosen for their ability to imperceptibly embed a large payload size in a visualization.

They both use a single color channel for embedding with a minimal pixel selection process. For LSB, pixels are selected from a given channel by a pseudorandom number generator whose seed is a key for embedder-extractor synchronization. For each pixel, the lowest k bits are replaced with the next k bits of the message. We found 4 bits per pixel to be a good trade-off between capacity and perceptibility. For HS, pixels are selected from a given channel based on histogram peaks and troughs. In this work we use the single peak-trough pair implementation.

The transform-space methods are Solanki's Entropy Thresholding [46] (ET) and Li and Lee's Pre-insertion Code [28] (PIC). Both methods use DCT and were chosen to showcase their potential resilience to JPEG compression. The ET method was designed for grayscale images. We opted to use the luminance (Y) channel for ET after converting to YCbCr color space. An RGB color channel may not have enough high frequency regions needed for data hiding. Using the luminance channel gives access to the overall highlight and shadow in the visualization, regardless of color.

PIC is an image-in-image watermarking technique, also designed for grayscale images. We chose this method to test whether an image watermark data can be generally replaced with an arbitrary bit sequence. We attempted using the luminance channel for embedding, but found that the color space conversion from YCbCr back to RGB would alter the payload, so we used a color channel instead for embedding.

Table 1 shows a sample of the set of images used for testing, showcasing the difference in datasets, visualization type, and rendering parameters that were modulated. We tested volume, isosurface, molecular dynamics, and streamline visualizations. All surface-based renders were performed 10 times, using a different specular component $k_s \in [0, 1)$ in steps of 0.1. Specular components $k_s \in [0.1, 0.3] \cup [0.5, 0.9]$ were omitted from Table 1 for space.

The datasets used in the image suite in Table 1 were chosen to be representative of common scientific visualization use cases. In these varying cases, it is important for scientists to retain accurate information and documentation about the visualizations. Examples of such crucial information include the specific transfer function used to highlight features in volume rendering a simulation time step (heptane, magnetic, and supernova); simulation state information, which may be stochastic in nature in e.g., a molecular dynamics simulation (CroV virus); and seeding information for streamlines and pathlines in turbulent flow simulations (tornado). These data are important not just for reproducibility of the visualization itself, but also for the entire scientific workflow in its related domain. These data may range from a few bytes, e.g., for storing isovalues, to kilobytes, e.g., for storing a script that generates streamlines from turbulent flow data.

Based on our evaluation, visualizations are a difficult target for information embedding. A single method is unable to

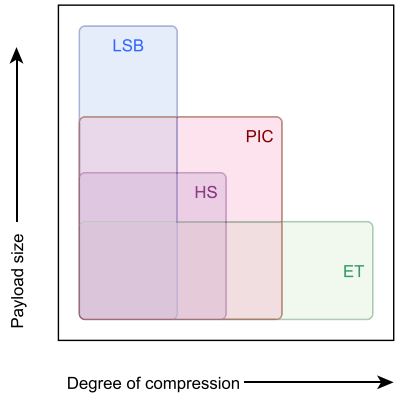


Fig. 5. The 2-dimensional payload size versus resilience space, illustrated with resilience to degree of compression. Payload size ranges from tokens and keys to full scripts. Degree of compression ranges from bitmap to JPEG quality 95. The theoretic zones for the chosen algorithms are shown here, drawn based on our testing and the original description of the algorithm. The "best of both worlds" area in the top right is difficult to obtain due to the trade-off in resiliency and payload size.

cover all use cases, but with informed choices, information can be reliably embedded. Some visualizations have a much higher capacity than others. Simple and practical design changes can be used to increase a visualization's capacity for payload. We have created a set of guidelines for background color and reduction of high-transparency features.

5.1 Classification by Payload Size versus Resilience

Maximizing payload size and maximizing resilience are our primary performance metrics. The four algorithms reside in different regions of the 2D payload-resilience space, as shown in Fig. 5.

Test Framework. To analyze the algorithms in the 2D space, we devised a testing framework to measure algorithm success rates, i.e., whether an algorithm can repeatably embed and extract payload data of a given size with a given compression level. Given an image from the test pool, we embedded and extracted 10 times with randomized payload data. If the extracted payload was identical to the data originally embedded, the attempt was successful.

Since the image-space methods have a relaxed pixel selection process, we could test with any image from our test pool. These methods tended to succeed at all 10 attempts, or fail at all 10. The PIC method also has a relaxed selection process. Every block in the image is used. The specific coefficients used per block could be customized. However, this method suffered from insertions and deletions at the encoder, and was unable to extract the payload correctly from any image. Although the ET method had a more stringent entropy-based selection process, the algorithm has a stochastic nature explained below. To accommodate this characteristic, an algorithm was considered successful if it correctly extracted the payload at least 8 out of 10 attempts.

Robust Embedding with ET Method. Embedding with the ET method has a stochastic nature due to the algorithm's query for blocks that remain above the chosen entropy threshold T. Suppose a binary subsequence b of the payload is taken. A DCT block from the image may have entropy $E \geq T$. Once b is embedded, the block's entropy is recalculated as E_b. E_b may or may not remain $\geq T$; if it is, the extractor should read it, otherwise the extractor will ignore it. However if $E_b \geq T$ after embedding but is altered during compression and drops below T, it may still be ignored by the extractor. This effect is entirely dependent on b, whose elements are hard to predict.

Hyperparameter Tuning with ET Method. An additional consideration for the ET method was the hyperparameters used when embedding and extracting: the quality level Q for compression resilience, the entropy threshold T for candidate blocks, and the number of bits B to embed per block. The exact value to use for a (Q, T, B) tuple depends on the image contents. If there is too little high frequency information, no value for (Q, T, B) can be successful.

We tested quality level $Q = 95$, in particular because most JPEG encoder implementations expose only the quality factor knob to the user, and most default the quality to 95. We also do not expect visualizations to be compressed beyond quality 90, as this would cause compression artifacts in the visualization. We tested with $T = [10000, 250000]$ in increments of 10000, $B = [1, 63]$, and payload sizes: 16, 32, 128, and 1024 bytes.

We used the heptane isosurface, CroV giant virus pseudo-atomic model, and CM1 tornado streamlines visualizations shown in Table 1. We ran preliminary tests to find optimal specular lighting conditions for the ET method. We found that a specular component $k_s = 0.4$ had the most success. This amount of specular lighting provided enough contrast in highlights and shadows across the surface for information hiding. The heptane surface with 40 percent specular lighting is shown in Fig. 4d. We also found that embedding with the JPEG 75 quantize matrix [22] improves reliability to counteract blocks that are less reliable for data embedding. For each test, the image was saved as JPEG quality 95 before extraction.

Fig. 6 shows success rates for extracting from the three visualizations with a payload size of 16 bytes. 32 byte payloads were less successful, and 128 and 1024 byte payloads failed nearly every time, due to insufficient eligible blocks for embedding. We note 16 bytes should be enough for the target use case of embedding authentication tokens or database keys.

Each visualization type has different success rates under different parameters. These success patterns heavily depend on the contents of the image, and would change if viewpoint and lighting conditions change. The giant virus visualization tended to work for higher bits per block values. This means that only the first blocks with entropy $> T$ were reliable for data embedding. Another possibility is that blocks containing a black carbon atom suffered from integer underflow during embedding, leading to white pixels. We discuss this phenomenon in more detail in Section 5.3.1.

Despite varying success patterns, there is a small common ground in mid-range values for T and B in our test images. We chose $T = 140000$ and $B = 40$ as the heuristic for all subsequent uses of the ET method.

Payload-Resilience Space Classification. Table 2 shows the success rate of the four algorithms together. For the ET method, we used the parameters discussed above. These algorithms were able to repeatably extract bit-for-bit the payload that was embedded within them at their respective positions in the payload-resiliency space. Note that PIC was

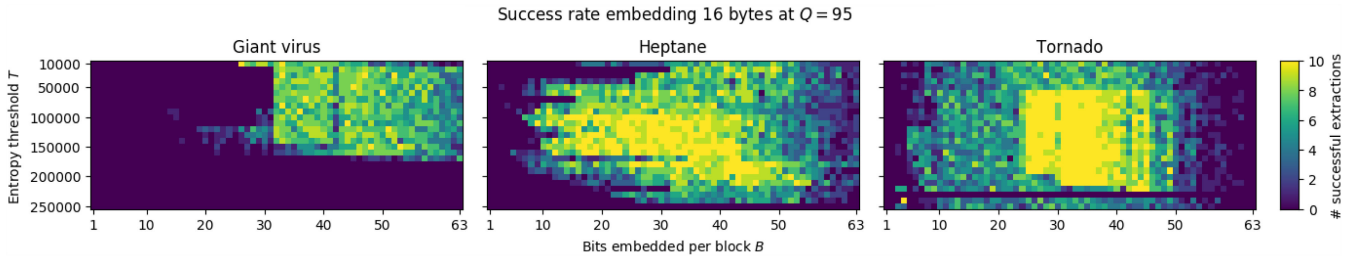


Fig. 6. Entropy Threshold method parameter search, showing successful embedding and extracting on the CroV giant virus pseudo-atomic model, heptane gas simulation surface, and CM1 tornado streamlines visualizations embedded with $Q = 75$ and saved with $Q = 95$. Each row in the tables represents an entropy threshold and each column represents a number of bits per block for embedding. Elements are colored by the number of successful extractions out of 10 iterations. Each visualization type exhibits quite different behavior in terms of successful parameters. From these results, we chose to use $T = 140000$ and $B = 40$ for embedding data with this method.

unable to successfully extract data as it suffered from insertions and deletions at the encoder.

The trade-off between payload size and resiliency can be clearly seen in Table 2. We recommend LSB if a full script is to be embedded, at the cost of resiliency. Any self-describing visualization using LSB must remain either a bitmap or PNG for the payload to survive. When using LSB, we recommend using $k = 4$ bits per pixel for payload density. This allows a maximum of 128 KB of embedded information in a 512×512 image, while introducing a maximum per-pixel brightness increase of $2^k = 16$ in the chosen color channel. This presents a good trade-off between data capacity and perceptibility. In our testing, higher values for k began to visibly show color noise (e.g., Fig. 2a uses $k = 6$). Embedding within the blue color channel is preferable due to the human visual system's weakness in detecting brightness changes in blue [72]. When embedding auth tokens or keys, we recommend the ET method, with the benefit of resilience to JPEG compression. We recommend using the $T = 140000$ and $B = 40$ values found above in our parameter search.

Narrowing Down Methods. PIC was unable to successfully extract data from images as it suffered from insertions and deletions at the encoder. We additionally tested with RS-coding to combat this, but the number of bits added to the extracted message or deleted from the embedded message defeated the encoding scheme. We did not test further with PIC. HS did successfully embed and extract information from various visualizations. However, HS has limited usefulness as it is surpassed by both LSB and ET in both axes of the two-dimensional space, as shown in Table 2.

TABLE 2
Suggested Algorithms for a Desired Payload Size and Degree of Compression, Based on Our Testing Framework

		Compression		
		BMP	PNG	JPEG $Q = 95$
Payload size (bytes)	64K	LSB	LSB	–
	8K	LSB	LSB	–
	1K	LSB	LSB	–
	128	LSB, HS	LSB, HS	–
	32	LSB, HS, ET	LSB, HS, ET	ET
	16	LSB, HS, ET	LSB, HS, ET	ET

These algorithms can extract payload from watermarked visualizations with bit-for-bit accuracy. There is a clear trade-off between payload size and resiliency. ET should be used for resiliency to JPEG, at the cost of payload size. On the other hand, LSB should be used when embedding full scripts or documentation, but requires the image stay in BMP or PNG format only.

5.2 Algorithmic Performance Benchmarks

We tested the ET and LSB methods for computing efficiency. We measured the average time to embed and extract over 10 iterations with varying payload sizes and image resolutions. The payload sizes tested were 16, 32, 64, and 128 bytes. LSB was also tested with 256, 512, and 1024 bytes. Test image resolutions were 512^2, 1024^2, and 2048^2.

We tested ET with JPEG $Q = 95$, entropy threshold $T = 140000$, and $B = 40$ bits per block. These values were determined based on the parameter space search performed above to perform well on multiple images. The LSB method was tested using the red color channel and $k = 4$ bits per pixel. The exact choice of color channel has no performance impact; in this case we choose red for visibility during testing. Using $k = 4$ bits per pixel provides a good trade-off between capacity and perceptibility, but does not affect performance as we are embedding the same number of bits regardless of k. All tests were performed on a single threaded process on a machine with 2x Intel Xeon E5-2650 v4 CPU and 128 GB of memory.

Fig. 7 shows the averaged embedding times for each method, payload size, and image resolution on the top, and corresponding extraction times on the bottom. The average time to embed and extract is negligibly affected by the payload size for both methods. This means that the underlying data hiding mechanism is not a computing bottleneck for self-describing visualizations.

Instead, image resolution plays a much larger role in performance. Both methods performed 3-4x slower when resolution was doubled. For the LSB method this is due to larger matrices to isolate and merge after embedding. For the ET method, there are many more 8×8 blocks to transform and compare. Blocks with entropy $> T$ may be fewer in number and/or scattered throughout the image. This leads to longer linear scans through the image. Hence, the main consideration for self-describing visualizations is the resolution, as it increases time to find data hiding regions.

5.3 Visualization Design Guidelines

We propose some guidelines for visualizations to be watermarked. These guidelines apply mainly for the ET method, as it performs a more complex pixel selection process. In

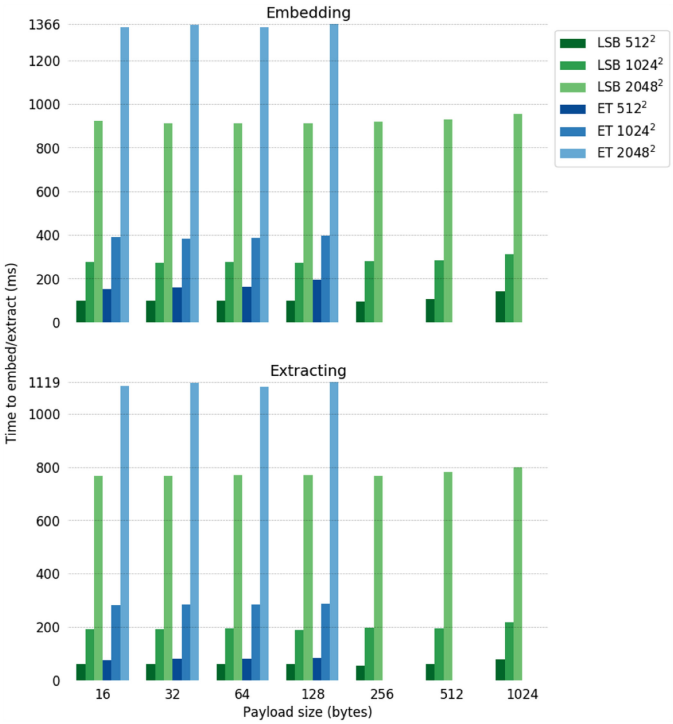


Fig. 7. Time to embed (top) and extract (bottom) with LSB and ET methods for varying payload sizes and image resolutions. Payload size has a negligible effect on performance. Instead, resolution plays a much larger role. Both methods perform 4x slower as image size doubles.

contrast, the LSB method is not image adaptive and thus is subject to fewer restrictions.

5.3.1 Minimizing Background and Black Pixels

Background-only regions of a visualization should be minimized. Visualizations should use dark gray colors (e.g., #080808) over pure black (#000000) whenever possible.

Coefficients in DCT space describe frequency information. Regions filled with a flat color contain only a DC coefficient. These regions have 0 entropy due to zero-value AC coefficients and are unsuitable for embedding.

Regions around the outer edge of a feature may have blocks containing feature and background. Luminance differences are often large enough at these edges that these regions are chosen for embedding. Embedded data may then cause visible artifacts in the flat background region, as shown in Fig. 8. Minimizing background-only regions increases the available textured regions for embedding data.

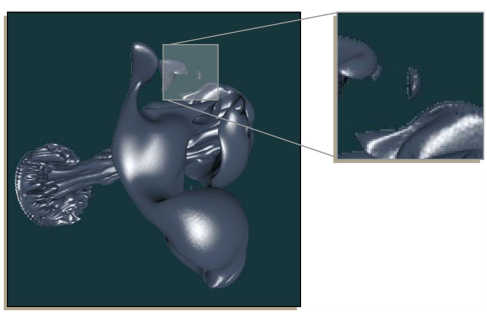


Fig. 8. A surface render of the heptane gas dataset watermarked with the ET method. Highlighted regions contain flat background which displays compression-like artifacts from the embedding process.

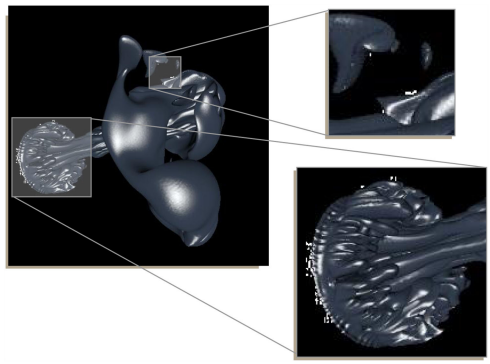


Fig. 9. A visualization with a black background watermarked by the ET method. Black regions may have underflow problems when the image is brought back to RGB space after data embedding, leading to noise.

In the special case of pure black, *all* frequency components are 0, including the DC component. In our extensive testing, we found this can lead to unsigned integer underflow noise when inverse transforming. The effect of this is that some pixels turn from black to white, as shown in the boundary regions of Fig. 9. Automatic color space compression may help with this. That is, compressing the full color range into a slightly smaller range, starting above black to remove all black pixels. However shadows on a surface may not be fully black and would not benefit from color space compression. The colormap used for surfaces and volumes may be misrepreseted after color space compression. In general, only the background should be regarded for fully black pixels to avoid altering the visualization. These stipulations do not apply to LSB, however, as it does not perform any image adaptive checks on the image.

5.3.2 Rendering Style

To increase entropy, volume rendered visualizations should have as high opacity for a feature as possible, or use surface rendering.

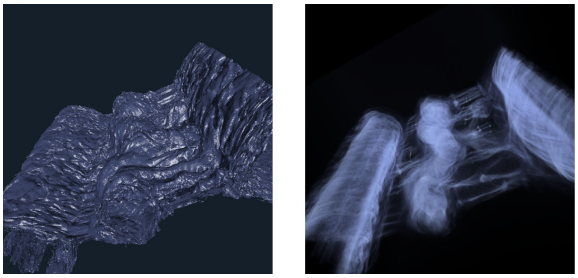
Volume renders with high transparency regions contain large expanses of low frequency luminance information. This is not suitable for watermarking with the ET method. Highly opaque volume features with lighting and shadow will contain higher luminance frequency. Surfaces with specular shading and ambient occlusion contain a better distribution of higher frequencies for hiding data.

For example, Fig. 10 shows the magnetic dataset rendered as a volume and surface with specular component $k_s = 0.5$. Using Equation (1) to calculate entropy on blocks of the images, the average non-zero entropy for the volume render is 5127.71, while it is 45913.47 for the surface render. This means there are many more candidate regions in the surface visualization.

Almost no highly transparent volume visualizations we tested worked well with the ET method. Often, only a single edge along a feature would have adequate frequency information for embedding. The LSB method had no issues with volume visualizations with its non-image-adaptive pixel selection process.

6 Sample Implementations

Embedding information in visualizations works directly on framebuffer content from the renderer. This makes



(a) (b)

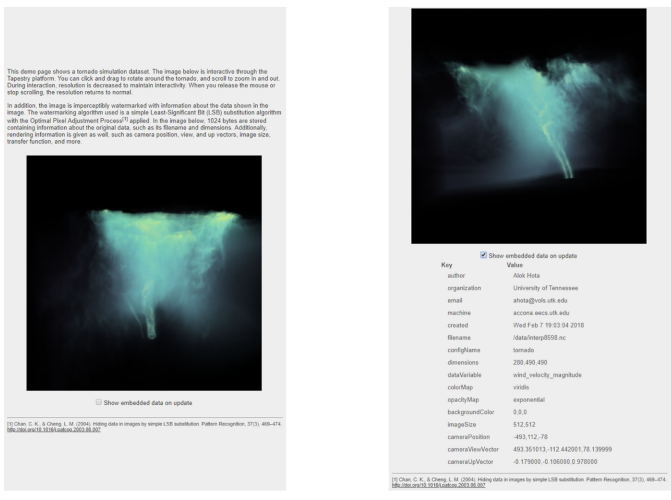
Fig. 10. A surface and volume render of the magnetic reconnection data. The surface contains much more high frequency luminance information due to specular lighting and ambient occlusion. This allows transform-space methods to work more effectively.

integration of embedding very straightforward regardless of the method used for embedder and extractor. As shown in Section 5.2, embedding and extraction times are 100-200 ms for the LSB and ET methods at 512^2 resolution. This makes visualization embedding appropriate for low to medium interactivity applications, but not ones requiring real-time performance.

In this section we discuss two sample implementations of self-describing visualizations. The first shows that embedding may be done as part of the render process. Watermarking at creation time provides a larger context for the visualization when viewed. The second shows how self-describing visualizations can be used to enhance documents. In both cases, the specific watermarking method used is flexible as it is a choice made at the application level.

6.1 Embedding in Rendering Pipeline

In this sample implementation, we show how a watermark can be embedded as a post-processing step in the rendering process. We used the OSPRay rendering engine [73] to generate images on the fly. We used LSB to embed 1,024 bytes of data in images. The LSB method can embed 1,024 bytes in 140 ms based on our testing. We found that OSPRay could



(a) (b)

Fig. 11. Watermarking can be performed at render time as a post-process filter. (a) screenshot shows a demo page with an OSPRay-rendered visualization of a tornado dataset using a web-based rendering service. Embedded data can be extracted as the image updates, as shown in (b).

(a) (b)

Fig. 12. This document as viewed in a sample web-based PDF reader. In (a), the yellow border indicates an image containing a valid watermark registered with the database. Clicking on any of the highlighted images in the paper shows an overlay containing the information from the database, (b). This enables immediate dissemination of visualizations in scientific papers.

render the tornado dataset at 512^2 resolution in 5-15 ms depending on viewpoint. This results in self-describing visualizations returning in under 150 ms.

OSPRay renders can also be made available via web services, such as Tapestry [61]. Thus self-describing visualizations are easily made available to the public for dissemination. Fig. 11 shows our sample implementation of post-process embedding. The embedded data contains author, contact, and dataset information along with rendering parameters, formatted as key-value pairs, shown in Fig. 11b. These key-value pairs form a JSON configuration format that can be used to regenerate a visualization in Tapestry.

6.2 Auto-Extractor in PDFs for Self-Describing Visualizations

In this sample we show how extracting the watermark data can be used to enhance the visualization within the context of a publication. There are a wealth of tools supporting the analysis of PDF files, such as PDF.js [74] for web-based viewing, and PDFMiner [75] for scripted analysis. Using these two libraries, we created a simple PDF web viewer. Users can upload a PDF to the web viewer. The PDF file is sent to a server which parses images from the document and attempts to extract watermarks.

To verify that the blind extractor retrieved valid data, the server performs a Redis [76] database query with the data as a key. If the key is valid, the data is returned. The PDF is rendered in the web browser with watermarked images given a yellow border. These images can be clicked on to show the associated data.

Fig. 12 shows two screenshots of the application in use on this document, which contains some watermarked images. We used 16-byte identifiers as keys to the database, embedded using the ET method with the hyperparameters found in our parameter search.

This application showcases how on-the-fly watermark extraction can add contextual information to visualizations in scientific publications. While reading, the audience can immediately verify the authenticity of a visualization based on the data returned, as well as understand how the visualization was created.

7 Discussion

Parameter Tuning. We present the above as generalized guidelines and sample implementations of self-describing visualizations. There are many parameters to tune as part of the optimization process. While a full, exhaustive

delineation of optimal configuration is out of the scope of this paper, herein we note that parameter choices in this work are empirical based on experiments and evaluation, as opposed to being driven by heuristics.

Error-Correcting Codes (ECC). It's likely to consider ECC as a topic related to watermarking. We note, however, that the overlap between these two areas is insignificant. Using the well-known ECC algorithm Reed-Solomon ECC as a reference, the primary use case is to defend against erasures, when there is a guaranteed length of data bits (i.e., data storage) [56]. In contrast, the main attacks we face in this work, because of our need for blind embedders and extractors to make visualizations self-describing, are instead insertions and deletions. These motivating needs and design constraints have little overlap.

In addition, we conducted a preliminary experiment using the RS(255, 223) coding scheme from [56]. Specifically, we RS-encoded provenance information before embedding it via watermarking. The results were worse than without using RS encoding. We do not know of the reasons conclusively. We suspect it has to do with insertions and deletions introduced in the extracted data caused by compression. We also note that RS(255,223) encoding introduces a reduction in effective payload size, as 32 of every 255 bits of available capacity is used for parity bits. We would like to investigate further in future work.

Data Documentation. Identifying datasets is a crucial element of self-describing visualization. While we used data path as identifier in this work, we believe the best solution to allow widespread adoption of self-describing datasets is usage of the existing Digital Object Identifier (DOI) system for tracking datasets. This would allow a scientist to simply embed an ID as the dataset identifier, which can then be referenced publicly by any audience member. This would facilitate publicly available datasets and provide a method for tracking visualizations in various scientific domains.

8 Conclusion

In this work, we showed that digital image watermarking techniques can be used to embed provenance information into scientific visualizations. We classified 34 watermarking algorithms from the literature based on requirements in visualization and recommend two useful techniques for creating self-describing visualizations–ET and LSB with Optimal Pixel Adjustment Process. These methods target use cases in visualizations ranging from embedding tokens to full scripts and configurations. We also developed a set of guidelines for optimizing payload capacity of visualizations based on our analysis of each embedding method's performance.

We demonstrated how self-describing visualizations can be used with two sample application implementations. Watermark embedding can be used as a post-process filter in the rendering pipelines, and can be extracted on the fly from PDFs containing self-describing visualizations. We believe self-describing visualizations can be easily integrated and used in conjunction with existing provenance systems, such as ParaView's lookmarks or VisIt's session files, and VisTrails.

For future work, we are also interested in extending our classifications and recommendations beyond images to videos. Provenance information could be embedded on a per-frame basis, or as part of the keyframe in video encoding schemes like MP4. We would also like to extend our work to information visualization, which exhibits vastly different characteristics to scientific visualizations.

Acknowledgments

The authors are supported in part by NSF Award CNS-1629890, Intel Parallel Computing Center (IPCC) at the Joint Institute of Computational Science of University of Tennessee, and the Engineering Research Center Program of the National Science Foundation and the Department of Energy under NSF Award Number EEC-1041877.

References

[1] E. T. Stanton and W. P. Kegelmeyer, "Creating and managing lookmarks in ParaView," in *Proc. IEEE Symp. Inf. Vis.*, 2004, pp. p19–p19.
[2] C. T. Silva, J. Freire, and S. P. Callahan, "Provenance for visualizations: Reproducibility and beyond," *Comput. Sci. Eng.*, vol. 9, no. 5, pp. 82–89, Sep. 2007.
[3] M. Raji, A. Hota, R. Sisneros, P. Messmer, and J. Huang, "Photo-guided exploration of volume data features," in *Proc. Eurographics Symp. Parallel Graph. Vis.*, 2017, pp. 31–39.
[4] "Further confirmation needed," *Nature Biotechnology, editorial*, vol. 30, no. 9, p. 806, Sep. 2012.
[5] "Reality check on reproducibility," *Nature*, vol. 533, no. 7604, pp. 437–437, May 2016.
[6] R. D. Peng, "Reproducible research in computational science," *Sci.*, vol. 334, no. 6060, pp. 1226–1227, Dec. 2011.
[7] J. Crocker and M. L. Cooper, "Addressing scientific fraud," *Sci.*, vol. 334, no. 6060, pp. 1182–1182, Dec. 2011.
[8] J. Kurose, "NSF 17–022: Dear colleague letter: Encouraging reproducibility in computing and communications research," *Nat. Sci. Found.*, 21 Oct. 2016. [Online]. Available: https://www.nsf.gov/pubs/2017/nsf17022/nsf17022.jsp
[9] D. Shands, "Big data regional innovation hubs: Establishing spokes to advance big data applications," *Nat. Sci. Found.*, 16 Mar. 2017. [Online]. Available: https://www.nsf.gov/pubs/2017/nsf17546/nsf17546.htm
[10] J. M. Perkel, "Data visualization tools drive interactivity and reproducibility in online publishing," *Nature*, vol. 554, no. 7690, pp. 133–134, Jan. 2018.
[11] E. Ragan, et al., "Characterizing provenance in visualization and data analysis: An organizational framework of provenance types and purposes," *IEEE Trans. Vis. Comput. Graph.*, vol. 22, no. 1, pp. 31–40, Jan. 2016.
[12] L. Bavoil, et al., "VisTrails: Enabling interactive multiple-view visualizations," in *Proc. IEEE Vis.*, 2005, pp. 18–18.
[13] D. Koop, et al., "A provenance-based infrastructure to support the life cycle of executable papers," *Procedia Comput. Sci.*, vol. 4, pp. 648–657, 2011.
[14] B. Bauer, et al., "The ALPS project release 2.0: Open source software for strongly correlated systems," *J. Statistical Mech.: Theory Experiment*, vol. 2011, no. 05, May 2011, Art. no. P05001.
[15] *Exchangeable image file format for digital still cameras: Exif Version 2.31*, CIPA DC-008-2016, Camera & Imaging Products Association, Jul. 2016.
[16] *Graphic technology – Extensible metadata platform (XMP) – Part 1: Data model, serialization and core properties*, ISO 16684-1:2019, Adobe Systems, Apr. 2019.
[17] *Graphic technology – Extensible metadata platform (XMP) – Part 2: Description of XMP schemas using RELAX NG*," ISO 16684-2:2014, Adobe Systems, Dec. 2014.
[18] Y.-C. Hou and P.-M Chen, "An asymmetric watermarking scheme based on visual cryptography," in *Proc. 5th Int. Conf. Signal Process.*, 2000, pp. 2–5.
[19] N. Bartlow, et al., "Protecting iris images through asymmetric digital watermarking," in *Proc. IEEE Workshop Autom. Identification Adv. Technol.*, 2007, pp. 192–197.
[20] H. Qi, "Human visual system based adaptive digital image watermarking," Master's thesis, School of Information Technology and Engineering, University of Ottawa, 2006.

[21] M. Barni and F. Bartolini, *Watermarking Systems Engineering: Enabling Digital Assets Security and Other Applications*. Boca Raton, FL, USA: CRC Press, 2004.

[22] G. K. Wallace, "The JPEG still picture compression standard," *IEEE Trans. Consumer Electron.*, vol. 38, no. 1, pp. xviii–xxxiv, Feb. 1992.

[23] W. Bender, et al., "Techniques for data hiding," *IBM Syst. J.*, vol. 35, no. 3.4, pp. 313–336, 1996.

[24] C.-Y. Lin, et al., "Rotation, scale, and translation resilient watermarking for images," *IEEE Trans. Image Process.*, vol. 10, no. 5, pp. 767–782, May 2001.

[25] I. J. Cox, et al., *Digital Watermarking and Steganography*, 2nd ed. Burlington, MA, USA: Morgan Kaufman Publishers Inc., 2008.

[26] V. M. Potdar, S. Han, and E. Chang, "A survey of digital image watermarking techniques," in *Proc. 3rd IEEE Int. Conf. Ind. Informat.*, 2005, vol. 7, pp. 709–716.

[27] M. Kutter and F. A. P. Petitcolas, "Fair benchmark for image watermarking systems," P. W. Wong and E. J. Delp III, Eds., Apr. 1999, pp. 226–239. [Online]. Available: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=979988

[28] J.-S. Lee and B. Li, "Self-recognized image protection technique that resists large-scale cropping," *IEEE MultiMedia*, vol. 21, no. 1, pp. 60–73, Jan.–Mar. 2014.

[29] J. M. Shieh, D. C. Lou, and M. C. Chang, "A semi-blind digital watermarking scheme based on singular value decomposition," *Comput. Standards Interfaces*, vol. 28, no. 4, pp. 428–440, 2006.

[30] X. Kang, et al., "A DWT-DFT composite watermarking scheme robust to both affine transform and JPEG compression," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 8, pp. 776–786, Aug. 2003.

[31] N. M. Makbol and B. E. Khoo, "Robust blind image watermarking scheme based on redundant discrete wavelet transform and singular value decomposition," *AEU-Int. J. Electron. Commun.*, vol. 67, no. 2, pp. 102–112, 2013.

[32] P. Bao, and M. Xiaohu, "Image adaptive watermarking using wavelet domain singular value decomposition," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 1, pp. 96–102, Jan. 2005.

[33] J. J. K. Ò. Ruanaidh and T. Pun, "Rotation, scale and translation invariant spread spectrum digital image watermarking," *Signal Process.*, vol. 66, no. 3, pp. 303–317, 1998.

[34] V. Solachidis and I. Pitas, "Circularly symmetric watermark embedding in 2-D DFT domain," *IEEE Trans. Image Process.*, vol. 10, no. 11, pp. 1741–1753, Nov. 2001.

[35] X. Y. Wang, et al., "A robust blind color image watermarking in quaternion fourier transform domain," *J. Syst. Softw.*, vol. 86, no. 2, pp. 255–277, 2013.

[36] F.-M. Tsai and W.-L. Hsue, "Image watermarking based on various discrete fractional fourier transforms," in *Proc. 13th Int. Workshop Digital-Forensics Watermarking*, 2015, vol. 9023, pp. 135–144.

[37] M. Urvoy, D. Goudia, and F. Autrusseau, "Perceptual DFT watermarking with improved detection and robustness to geometrical distortions," *IEEE Trans. Inf. Forensics Secur.*, vol. 9, no. 7, pp. 1108–1119, Jul. 2014.

[38] K. Solanki, et al., "'Print and Scan' resilient data hiding in images," *IEEE Trans. Inf. Forensics Secur.*, vol. 1, no. 4, pp. 464–478, Dec. 2006.

[39] M.-S. Hsieh, D.-C. Tseng, and Y.-H. Huang, "Hiding digital watermarks using multiresolution wavelet transform," *IEEE Trans. Ind. Electron.*, vol. 48, no. 5, pp. 875–882, Oct. 2001.

[40] H. M. Al-Otum and N. A. Samara, "A robust blind color image watermarking based on wavelet-tree bit host difference selection," *Signal Process.*, vol. 90, no. 8, pp. 2498–2512, 2010.

[41] D. Kundur and D. Hatzinakos, "Digital watermarking using multiresolution wavelet decomposition," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 1998, vol. 5, pp. 2969–2972.

[42] S. Xiang and Y. Wang, "Distortion-free robust reversible watermarking by modifying and recording IWT means of image blocks," in *Proc. 14th Int. Workshop Digital-Forensics Watermarking*, 2016, vol. 9023, pp. 337–349.

[43] M. Rabbani and R. Joshi, "An overview of the JPEG 2000 still image compression standard, *Signal Process. Image Commun.*, vol. 17, no. 1, pp. 3–48, 2002.

[44] D. Le Gall, "MPEG: A video compression standard for multimedia applications," *Commun. ACM*, vol. 34, no. 4, pp. 46–58, 1991.

[45] Y. Zhou and J. Liu, "Blind watermarking algorithm based on DCT for color images," in *Proc. 2nd Int. Congress Image Signal Process.*, 2009, pp. 2–4.

[46] K. M. Solanki, "Multimedia data hiding: From fundamental issues to practical techniques," PhD dissertation, Dept. of Electrical and Computer Eng., Univ. of California Santa Barbara, 2005.

[47] H. Zhang and X.-Q. Li, "Geometrically invariant image blind watermarking based on speeded-up robust features and DCT transform," in *Proc. Int. Workshop Digital Forensics Watermarking*, 2013, pp. 111–119.

[48] Z. Haitao, Q. Chun, and G. Xiaochuan, "Low luminance smooth blocks based watermarking scheme in DCT domain," in *Proc. Int. Conf. Commun. Circuits Syst.*, Jun. 2006, pp. 19–23.

[49] K. Solanki, et al., "Robust image-adaptive data hiding using erasure and error correction," *IEEE Trans. Image Process.*, vol. 13, no. 12, pp. 1627–1639, Dec. 2004.

[50] P. H. W. Wong, O. C. Au, and Y. M. Yeung, "A novel blind multiple watermarking technique for images," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 8, pp. 813–830, Aug. 2003.

[51] R. G. van Schyndel, A. Z. Tirkel, and C. F. Osborne, "A digital watermark," in *Proc. 1st Int. Conf. Image Process.*, 1994, vol. 2, pp. 86–90.

[52] N. Dey, A. B. Roy, and S. Dey, "A novel approach of color image hiding using RGB color planes and DWT," *Int. J. Comput. Appl.*, vol. 36, no. 5, pp. 975–8887, 2011.

[53] D. Rawat and V. Bhandari, "A steganography technique for hiding image in an image using LSB method for 24 bit color image," *Int. J. Comput. Appl.*, vol. 64, no. 20, pp. 975–8887, 2013.

[54] I. J. Cox, et al., "Secure spread spectrum watermarking for multimedia," *IEEE Trans. Image Process.*, vol. 6, no. 12, pp. 1673–1687, Dec. 1997.

[55] J. R. Smith and B. O. Comiskey, "Modulation and information hiding in images," in *Proc. 1st Int. Workshop Inf. Hiding*, 1996, vol. 1174, pp. 207–226.

[56] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Ind. Appl. Math.*, vol. 8, no. 2, pp. 300–304, Jun. 1960.

[57] A. S. J. Helberg and H. C. Ferreira, "On multiple insertion/deletion correcting codes," *IEEE Trans. Inf. Theory*, vol. 48, no. 1, pp. 305–308, Jan. 2002.

[58] L. Schulman and D. Zuckerman, "Asymptotically good codes correcting insertions, deletions, and transpositions," *IEEE Trans. Inf. Theory*, vol. 45, no. 7, pp. 2552–2557, Nov. 1999.

[59] T. Le and H. Nguyen, "New multiple insertion-deletion correcting codes for non-binary alphabets," *CoRR*, 2015. [Online]. Available: http://arxiv.org/abs/1502.02727

[60] F. A. P. Petitcolas, R. J. Anderson, and M. G. Kuhn, "Information hiding-A survey," *Proc. IEEE*, vol. 87, no. 7, pp. 1062–1078, Jul. 1999. [Online]. Available: http://ieeexplore.ieee.org/document/771065/

[61] M. Raji, A. Hota, and J. Huang, "Scalable web-embedded volume rendering," in *Proc. IEEE 7th Symp. Large Data Anal. Vis.*, Oct. 2017, pp. 45–54.

[62] K. Moreland, "Why we use bad color maps and what you can do about it," in *Proc. Human Vis. Electron. Imag. (HVEI)*, doi: 10.2352/ISSN.2470-1173.2016.16.HVEI-133.

[63] K. Moreland, "Diverging color maps for scientific visualization," *Lecture Notes Comput. Sci.*, vol. 5876, no. PART 2, pp. 92–103, 2009.

[64] P. Tsai, Y. C. Hu, and H. L. Yeh, "Reversible image hiding scheme using predictive coding and histogram shifting," *Signal Process.*, vol. 89, no. 6, pp. 1129–1143, 2009.

[65] M. D. Swanson, Z. Bin, and A. H. Tewfik, "Transparent robust image watermarking," in *Proc. 3rd IEEE Int. Conf. Image Process.*, 1996, vol. 3, pp. 211–214.

[66] J. Mielikainen, "LSB matching revisited," *IEEE Signal Process. Lett.*, vol. 13, no. 5, pp. 285–287, May 2006.

[67] Z. Ni, et al., "Reversible data hiding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 3, pp. 354–362, Mar. 2006.

[68] D. He and Q. Sun, "A practical print-scan resilient watermarking scheme," in *Proc. IEEE Int. Conf. Image Process.*, 2005, pp. I–257.

[69] C. Podilchuk and W. Zeng, "Perceptual watermarking of still images," in *Proc. IEEE 1st Workshop Multimedia Signal Process.*, 1997, pp. 363–368.

[70] C. K. Chan and L. M. Cheng, "Hiding data in images by simple LSB substitution," *Pattern Recognit.*, vol. 37, no. 3, pp. 469–474, 2004.

[71] K. Muhammad, et al., "A novel magic LSB substitution method (M-LSB-SM) using multi-level encryption and achromatic component of an image," *Multimedia Tools Appl.*, vol. 75, no. 22, pp. 14 867–14 893, 2016.

[72] G. Wagner and R. M. Boynton, "Comparison of four methods of heterochromatic photometry," *J. Optical Soc. Amer.*, vol. 62, no. 12, pp. 1508–1515, 1972.
[73] I. Wald, et al., "OSPRay - A CPU ray tracing framework for scientific visualization," *IEEE Trans. Vis. Comput. Graph.*, vol. 23, no. 1, pp. 931–940, Jan. 2017.
[74] Mozilla Foundation, "PDF.js - A general-purpose, web standards-based platform for parsing and rendering PDFs," 2018. [Online]. Available: https://mozilla.github.io/pdf.js/.
[75] Y. Shinyama, "PDFMiner - Python PDF parser and analyzer," 2018. [Online]. Available: https://euske.github.io/pdfminer/.
[76] Salvatore Sanfilippo, "Redis," 2018. [Online]. Available: https://github.com/antirez/redis.

Alok Hota received the bachelor of science degree in computer science from Fisk University, the bachelor of engineering degree in computer engineering from Vanderbilt University, and the master of science degree in computer science from the University of Tennessee, Knoxville. This work was completed as part of his PhD dissertation with the University of Tennessee, Knoxville. He is presently technical staff at Intel. He is a student member of the IEEE.

Jian Huang received the BEng degree in electrical engineering from the Nanjing University of Posts and Telecom, China, in 1996, the dual MS degrees in computer science and in biomedical engineering, both from Ohio State University, in 1998, and the PhD degree in computer science from Ohio State University, in 2001. He is a professor with the Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville. His research focuses on data analytics, data visualization, and parallel, distributed and remote visualization systems. His research has been funded by National Science Foundation, Department of Energy, Department of Interior, NASA, UT-Battelle and Intel. He was a recipient of DOE Early Career Principal Investigator Award, in 2004. He is a senior member of the IEEE.