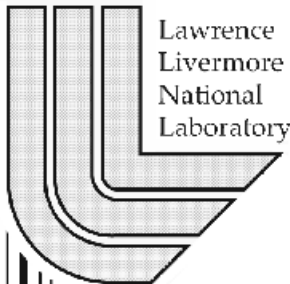


REVERSIBLE N -BIT TO N -BIT INTEGER HAAR-LIKE TRANSFORMS

Joshua G. Senecal
Mark A. Duchaineau
Kenneth I. Joy

This paper was accepted for publication and presentation at the 7th IASTED International Conference on Computer Graphics and Imaging, to be held August 16–18, 2004, in Kauai, Hawaii

U.S. Department of Energy



Lawrence
Livermore
National
Laboratory

August 4, 2004

DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

This is a preprint of a paper intended for publication in a journal or proceedings. Since changes may be made before publication, this preprint is made available with the understanding that it will not be cited or reproduced without the permission of the author.

This research was supported under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

REVERSIBLE N -BIT TO N -BIT INTEGER HAAR-LIKE TRANSFORMS

Joshua G. Senecal^{*‡}

Mark A. Duchaineau[†]

Kenneth I. Joy[‡]

^{*}Institute for Scientific Computing Research

[‡]Institute for Data Analysis and Visualization

[†]Center for Applied Scientific Computing

Computer Science Department

Lawrence Livermore National Laboratory

University of California, Davis

{senecall, duchaine}@llnl.gov

kijoy@ucdavis.edu

ABSTRACT

We introduce TLHaar, an n -bit to n -bit reversible transform similar to the S-transform. TLHaar uses lookup tables that approximate the S-transform, but reorder the coefficients so they fit into n bits. TLHaar is suited for lossless compression in fixed-width channels, such as digital video channels and graphics hardware frame buffers. Tests indicate that when the incoming image data has lines or hard edges TLHaar coefficients compress better than S-transform coefficients. For other types of image data TLHaar coefficients compress up to 2.5% worse than those of the S-transform, depending on the data and the compression method used.

KEY WORDS

Image Processing, Fixed-Width Transforms, Wavelets, Image Compression

1 Introduction

Integer wavelet transforms have what is termed *dynamic range expansion*. Simply put, dynamic range expansion means that the range over which wavelet coefficients can take their values is larger than that of the input domain. This usually requires that the number of bits required to hold the wavelet coefficients is greater than the number of bits required to hold the input data (for a discussion of dynamic range expansion and its effects see [1]). A common integer wavelet transform is the S-transform [2], which is an integer version of the Haar wavelet transform[3][§]. The Haar transform is defined by:

$$\tilde{H} = \frac{B - A}{\sqrt{2}} \quad (1)$$

$$\tilde{L} = \frac{A + B}{\sqrt{2}} \quad (2)$$

^{*}L-419, PO Box 808, Livermore, CA 94551, U.S.A., Tel: 925-422-3764, Fax: 925-422-7819

[†]L-561, PO Box 808, Livermore, CA 94551, U.S.A.

[‡]One Shields Ave, Davis, CA 95616, U.S.A.

[§]Throughout this paper \tilde{H} and \tilde{L} denote coefficients produced by Haar, \hat{H} and \hat{L} denote those produced by the S-transform, and H and L denote those produced by our method, TLHaar.

and the S-transform by:

$$\hat{H} = B - A \quad (3)$$

$$\hat{L} = \left\lfloor \frac{A + B}{2} \right\rfloor. \quad (4)$$

where A and B are two adjacent values to be transformed.

In the S-transform dynamic range expansion is due to the subtraction that occurs in the transform procedure—it is necessary to store a sign bit for all nonzero high-pass coefficients. These sign bits present some problems. First, there is raw data inflation: the number of bits required to store the transformed data is greater than that required to store the original data. Second, since modern computers store data in 8-bit chunks, an actual implementation that takes (for example) 8-bit inputs must store each 9-bit coefficient in a data type 16 bits wide, thus consuming twice the memory bandwidth. If this transform is being done in a fixed-width hardware environment and 16-bit values are unavailable, some data loss will result.

Referring to the Haar equations 1 and 2, if we remove the normalization by $\sqrt{2}$ we have the non-normalized Haar transform $\dot{H} = B - A$ and $\dot{L} = A + B$ (note that $\hat{H} = \dot{H}$). We see from figure 1 that the non-normalized Haar transform expands the domain by $\sqrt{2}$ and rotates it by 45 degrees. This gives new high- and low-pass values a range twice that of the original domain, as measured along the axes. It should be obvious that if the inputs to the non-normalized Haar transform are limited to an integer domain then the range will also contain only integers. Furthermore, the number of positions occupied in the range will be equal to the number occupied in the domain, with the range being organized in a lattice.

The S-transform takes advantage of the space in between entries in the range lattice, and “squashes” the non-normalized Haar range so that its low-pass values fall into the original domain. This squashing $\dot{L} \rightarrow \hat{L}$ is done by right-shifting \dot{L} one bit, thus eliminating the LSB. Because the LSB of \dot{H} and \dot{L} are identical, no information is lost during the squashing process—the LSB needed to reconstruct \dot{L} can be taken from \dot{H} —and the transform is completely reversible. While this effectively eliminates dy-

dynamic range expansion from the low-pass coefficients, the range of high-pass coefficients is still expanded.

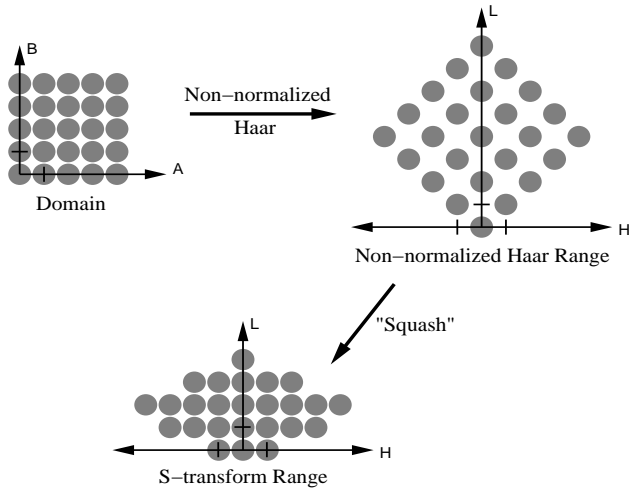


Figure 1. Domain and range of the non-normalized Haar and S transforms.

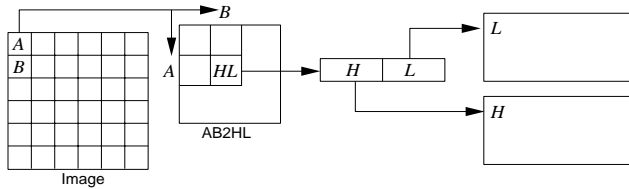


Figure 2. TLHaar transform process.

In [4] a wavelet transform for binary images (bilevel, 1 bit per pixel) that overcomes these problems is described. We present one solution to these problems in the more general greyscale case: the Table-Lookup Haar (TLHaar) method, a reversible n -bit to n -bit transform. Because it is a fixed-width transform TLHaar is particularly suited for lossless compression and use in environments with fixed-width channels, such as digital video and graphics hardware frame buffers. TLHaar uses two lookup tables (LUTs) called AB2HL and HL2AB, each of size 2^{2n} . These tables make the transform's range equal to that of the domain while preserving the coefficient magnitude ordering relationships of the Haar transform.

To evaluate TLHaar we assembled a suite of 8-bit image sets, with each set containing images of a particular type (bilevel, shaded line art, photographs, etc.). We then implemented and optimized TLHaar and the S-transform. We recorded their execution times and compressed the coefficients they produced with several coders. Our tests indicate that TLHaar is up to 44% faster than the S-transform, particularly when transforming data in large chunks. For data that have sharp edges, such as bilevel, line art, and computer generated images, coefficients generated by TLHaar compress better than S-transform. For other types of

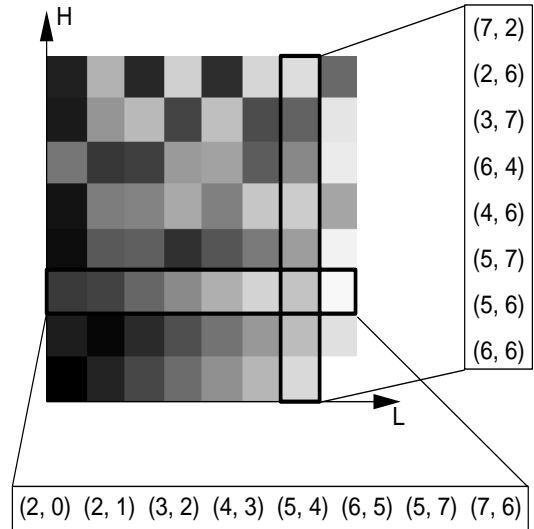


Figure 3. The HL2AB LUT for $n = 3$, showing a column and row breakout. Entries are (A, B) .

data the compression ratio is up to 2.5% worse, depending on the data and compression method used. Since TLHaar was developed to first address the problem of dynamic range expansion, we feel these compression ratios are acceptable.

2 Table-Lookup Haar

Table-Lookup Haar (TLHaar) performs a Haar-like transform, replacing the averaging and differencing steps with a single table lookup. TLHaar uses a set of two 2D lookup tables, called AB2HL and HL2AB. Each table contains 2^{2n} entries, with each entry being $2n$ bits wide, the upper and lower n bits each containing a value. AB2HL is used when performing a forward transform. It takes two n -bit data values A and B as indices and produces a $2n$ bit value, where the upper n bits are the high-pass value H and the lower n bits are the low-pass value L . HL2AB is used when reversing a transform, similarly converting (H, L) to (A, B) . The transform process is illustrated in figure 2.

2.1 Transform LUTs

The HL2AB table used in TLHaar is designed to mimic the true Haar transform by satisfying equivalent ordering relationships on the magnitude of the high-pass values and low-pass values within each respective row and column of the table.

$$\forall L : |\tilde{H}_i| \leq |\tilde{H}_j| \iff H_i \leq H_j \quad (5)$$

$$\forall H : \tilde{L}_i \leq \tilde{L}_j \iff L_i \leq L_j \quad (6)$$

For the transform table to be reversible there must be a 1:1 mapping between entries in the two tables. We therefore

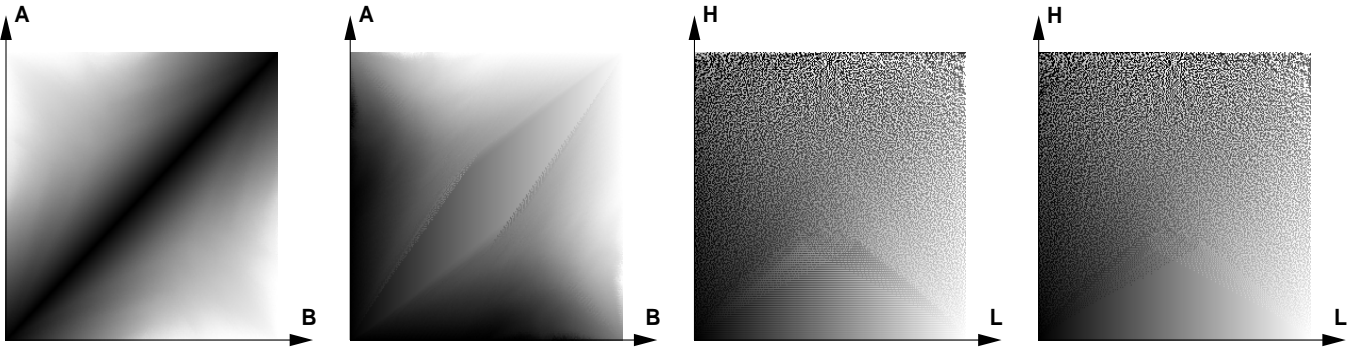


Figure 4. From left to right, the $(A, B) \rightarrow H$, $(A, B) \rightarrow L$, $(H, L) \rightarrow A$, and $(H, L) \rightarrow B$ transforms, as taken from the AB2HL and HL2AB tables. Black = 0, White = 255, (0,0) is the lower-left corner of each table.

initialize each with an identity transform: $AB2HL[i, j] = (i, j)$, $HL2AB[i, j] = (i, j)$. Note that because no entry has a value larger than 2^n dynamic range expansion cannot occur. We then rearrange the entries in HL2AB and AB2HL so properties 5 and 6 hold. We accomplish this via a sort of the LUTs according to the following pseudocode:

```
do {
  For each L Column in HL2AB
    Sort based on  $|(B - A)|$ 
  For each H Row in HL2AB
    Sort based on  $(A + B)$ 
} while (there was a swap)
```

During the sort process whenever a swap occurs in HL2AB the corresponding entries in AB2HL are also swapped. Figure 3 shows the HL2AB LUT for $n = 3$, with a column and row breakout for a particular table entry. Note how all column entries satisfy property 5 and all row entries satisfy property 6.

It was not clear beforehand that this sort would converge. At this time we do not have a general proof that the sort will always converge, however we tested this process of creating tables for values $2 \leq n \leq 12$, and our tests indicate that in all cases the sort converges. We are unable to test further since when $n > 12$ the tables become so large they are impractical. For $n = 13$ a single LUT will contain over 67 million entries and be 256 megabytes in size.

Figure 4 shows the $(A, B) \rightarrow (H, L)$ and $(H, L) \rightarrow (A, B)$ transforms from the AB2HL and HL2AB LUTs after sorting, when $n = 8$. From these we see that TLHaar performs the desired transform: for example, A and B that are close in value have an H close to zero as shown by the dark band on the diagonal of $(A, B) \rightarrow H$. Figure 5 gives the histogram of the “Lena” image before and after a TLHaar transform.

2.2 Sorted LUTs Are Not Unique

In section 2.1 we demonstrated the method used for creating the TLHaar LUTs, using a sort on an identity trans-

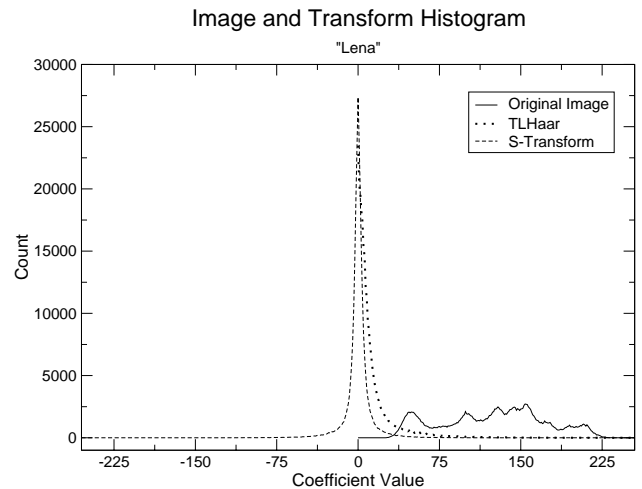


Figure 5. The histogram of the original pixels and the transform coefficients of the “Lena” image.

form. If the LUTs are initialized with a 1:1 mapping different from our original method, will the sort still result in the same transform LUTs? The answer to this question is no, implying that LUTs satisfying relationships 5 and 6 are not unique. As a test we permuted one of the initialized LUTs before sorting, by randomly swapping entries. The corresponding entries in the other LUT were also swapped, maintaining a 1:1 mapping. We then performed our sort on the permuted tables. The resulting transform tables were different from those of our original method. By varying the number of swaps and the seed to the pseudorandom number generator we produced tables that were similar to each other, but not identical.

3 TLHaar Implementation Optimizations

Because TLHaar operates on and produces n -bit data, when n is both a power of 2 and an integer size common in modern computer architectures (8-bit byte, 16-bit short integer, etc.) it is possible to store the low-pass and high-

pass values in arrays of that integer type. This allows us to implement and take better advantage of some special optimizations. Here we describe optimizations made for an implementation that operates on 8-bit images.

We first altered how we perform table lookups in a row transform. Since input values A and B are adjacent in memory, instead of reading A and B separately and indexing the AB2HL LUT with both (i.e. $HL = AB2HL[A][B]$) we cast the input array of bytes into an array of 16-bit short integers, and read A and B together as a single short AB . This allows us to perform a complete table lookup using fewer operations: $HL = AB2HL[AB]$.

We would like to use the above optimization when performing a transform in the column direction. The standard row transform operates on an image one row at a time, writing out the resulting low-pass values such that they are contiguous in the row direction. Thus a given image column is not contiguous in memory. To solve this when performing a row transform we transform two rows at a time. Given the k -th pair of pixels from rows i and $i+1$ we transform $A_i B_i$ and $A_{i+1} B_{i+1}$, and place L_i and L_{i+1} adjacent to each other in preparation for the column transform. The column transform can then proceed down columns in image space, but along adjacent memory locations. The idea behind the optimization is shown in figure 6. The right side shows the low-pass values ordered in memory.

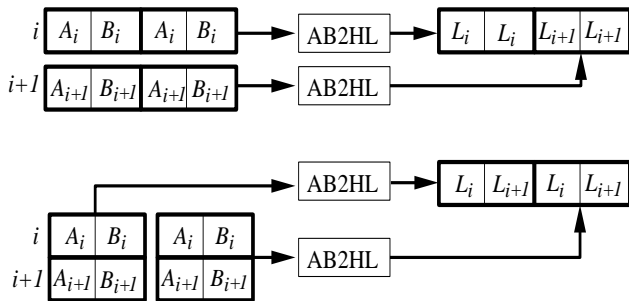


Figure 6. The TLHaar row transform data reordering optimization. The top shows the normal transform procedure, and the bottom the reordering procedure.

4 Results

To obtain the results in this paper we assembled a suite of 8-bit images. The images fall into the following categories:

- **BW_Lines.** A collection of 106 bilevel (black-and-white) line figures. Each figure is small, an example size being 108 by 110 pixels [5].
- **LineArt.** A set of 18 line art images. Each image has shading, gradient fills, and the like.
- **ObjectBank.** A set of 122 computer-rendered images of everyday objects [5].

- **MRI.** A set of 185 MRI scans. Each image is 256 by 256 pixels [6].
- **ccitt.** Eight of the standard bilevel (black-and-white) ccitt FAX test images [7]. Each is 1728 by 2376 pixels.
- **DB1_B.** A set of 80 fingerprint scans[8]. Each is 300 by 300 pixels.
- **DB2_B.** A set of 80 fingerprint scans. These are the same fingerprints as in DB1_B, but each image is 256 by 364 pixels, and the images have been processed to bring out details.
- **Photos.** A set of 29 photographic images [9, 10]. These include standard test images (such as “Lena”) and personal photographs from the lead author.
- **r2_slices.** A set of 221 randomly selected images extracted from the data produced by a Richtmeyer–Meshkov mixing simulation, described in [11].
- **Power2.** This is a selection of images from Photos, where each image is square and has an edge length that is a power of 2. These are used to evaluate the reordering method described in section 3.

In each table of results the column heading “% Gain” indicates gains obtained using TLHaar over the S-transform (positive percentage meaning TLHaar is better).

4.1 Execution Time

To compare execution times we implemented procedures to perform the S-transform and TLHaar transforms on 8-bit grayscale images, and optimized each separately. We also implemented versions of TLHaar and the S-transform that perform data reordering and operate on square images with edge lengths that are a power of 2. Timings were taken on a 550 MHz PowerBook G4 running MacOS 10.1.5. The time given is an average over 10 transform runs, where a run transforms all images in a particular category, and includes only the time taken to transform the image. Execution times are given in table 1.

4.2 Compressibility of Coefficients

As our main interest in developing TLHaar was to create a fixed-width transform we are not greatly concerned about how efficiently TLHaar’s coefficients compress compared to the S-transform. If the resulting compression ratio of TLHaar coefficients is within a percent or two of the rate obtained by the S-transform, we are content. To get a feel for how coefficients generated by TLHaar compress compared to those produced by the S-transform we transformed the test images and then compressed the results using three freely available compression programs: `gzip`¹,

¹<http://www.gzip.org>

Category	Transform Time (sec)		% Gain
	TLHaar	S-transform	
BW_Lines	0.06368	0.06540	2.63
LineArt	0.15289	0.27621	44.65
ObjectBank	0.90947	1.08220	15.96
MRI	0.45442	0.55188	17.66
MRI (reord)	0.38121	0.51650	26.19
ccitt	1.31669	1.96779	33.09
DB1_B	0.25860	0.36670	29.48
DB2_B	0.31501	0.35577	11.46
Photos	1.02017	1.34319	24.05
r2_slices	0.31479	0.39773	20.85
Power2	0.23322	0.31506	25.98
Power2 (reord)	0.20557	0.29665	30.70

Table 1. Our test image categories and their transform times. (reord) indicates execution time using the reordering method of section 3.

bzip², and an arithmetic coder available from Alistair Moffat³. We used binary and byte arithmetic encoding.

To gauge the effect of sign bits on the compressibility of S-transform coefficients we compressed them in two ways. In the first method the coefficient magnitudes were written as a stream of bytes and compressed, and the sign bit for each nonzero magnitude was appended uncompressed. In the second method coefficient magnitudes were written as a stream of bytes, and then a binary stream consisting of the sign bits of all nonzero magnitudes was appended. The combined stream was then compressed.

Due to lack of space we present in table 2 results only for the former method, as it presents the S-transform more favorably and the comparison to TLHaar is more fair. Generally when the latter method is used, the S-transform coefficients do not compress nearly as much as the TLHaar coefficients. With byte arithmetic coding the TLHaar coefficients always compress to a smaller size. Some example results in this case are TLHaar being 4.45% better in the Photos category and 7.23% better in the MRI category.

Results for TLHaar in table 2 are when using unpermuted tables, as described in section 2.1. We do not include the size of the transform LUTs in the TLHaar coefficient sizes. The tables are a static part of the transform process and are therefore known ahead of time, so in a coding application the tables do not need to be sent as part of the encoded data.

5 Conclusions and Future Research

The suitability of TLHaar depends primarily on the need for protecting data against loss. If data needs to be processed losslessly in a fixed-width environment, TLHaar—by virtue of its fixed-width nature alone—is superior to the S-transform.

²<http://sources.redhat.com/bzip2/>

³http://www.cs.mu.oz.au/~alistair/arith_coder/

From our timing tests it appears that TLHaar is faster than the S-transform, particularly when data reordering is implemented. The only case in which TLHaar’s speed is close to that of the S-transform is when the images are rather small, as is the case with the BW_Lines image set. Although the best case 44% speed increase in our tests may not appear to be a large gain, we point out that this speed gain is being taken against a wavelet transform that is already very fast. These speed results are promising, but further tests on different hardware platforms are needed to obtain a better indication of TLHaar’s execution time.

From our simple compression tests TLHaar coefficients appear to compress significantly more than S-transform coefficients when the data being processed is bilevel, or contains lines or hard edges (as are in the BW_Lines, LineArt, ObjectBank, ccitt, and DB2_B sets). For other classes of images the results are mixed, and vary depending on the coding method used. For example, when using gzip as the compressor, the TLHaar coefficients produced for the Photos category compress 0.78% worse than the S-transform coefficients. The gap widens to 2.43% worse when binary arithmetic coding is used. Conversely for the MRI data set gzip compresses TLHaar coefficients 1.78% worse than S-transform coefficients, but using byte arithmetic encoding this gap narrows to only 0.013% worse. Given that TLHaar was designed to operate in a fixed-width environment and the S-transform was not, we feel these compression rates are acceptable.

We do not expect our current implementation of TLHaar to be very useful for lossy compression. Future research will focus on developing variations of TLHaar that are more suitable for lossy techniques.

Other research will center around studying the LUTs in more detail. During the transform process each image type only touches a small percentage of the total entries in the LUT. These entries are often in clusters. It may therefore be possible to create LUTs for specific data types by optimizing only those parts of the LUT that the image touches.

6 Acknowledgements

We are grateful to Henrique Malvar, Aaron Kiely, Matthew Klimesh, and our reviewers for their helpful comments and suggestions.

Images in our test suite were provided by a variety of sources. In particular, the BW_Lines and ObjectBank images were obtained from Michael J. Tarr at Brown University [5]. The LineArt images are clip art downloaded through AppleWorks, from Apple Computer, Inc. Some images in the Photos category were obtained from the Signal and Image Processing Institute at the University of Southern California [9], and others from the Waterloo BragZone maintained by John Kominek [10]. MRI data was provided by the Department of Radiology at UCSD Medical Center, and the Vision List Imagery archive [6]. Fingerprint images were obtained from the web site of

Image	TLHaar	S-transform	% Gain	TLHaar	S-transform	% Gain
	gzip			bzip		
BW_Lines	360972	466731	22.66	391737	500173	21.68
LineArt	419235	528964	20.74	445387	515182	13.55
ObjectBank	3131056	3341803	6.31	3115625	3258474	4.38
MRI	5477859	5381934	-1.78	5281869	5190000	-1.77
ccitt	1157989	1557889	25.67	1013614	1454085	30.29
DB1_B	5016509	5086675	1.38	4973728	4853424	-2.48
DB2_B	5820744	6165921	5.60	6175512	6353409	2.80
Photos	15029496	14912472	-0.78	14963474	14651157	-2.13
r2_slices	777975	801756	2.97	785564	783836	-0.22
	Binary Arithmetic			Byte Arithmetic		
BW_Lines	391472	494289	20.80	377422	496202	23.94
LineArt	509478	627262	18.78	535573	677915	21.00
ObjectBank	3290576	3474223	5.29	3620160	3771556	4.01
MRI	5691930	5627898	-1.14	5408487	5407783	-0.013
ccitt	1199500	1652339	27.41	1288297	1969556	34.59
DB1_B	5673653	5346789	-6.11	4796648	4703212	-1.99
DB2_B	6415074	6803730	5.71	5679509	5946472	4.49
Photos	15234370	14873308	-2.43	14285797	14090810	-1.38
r2_slices	828160	850696	2.65	823509	834297	1.29

Table 2. Compressed category sizes (in bytes).

the Fingerprint Verification Competition 2000 [8]. We are grateful to all those who made data available.

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

Joshua Senecal's work was supported in part by a United States Department of Education Government Assistance in Areas of National Need (DOE-GAANN) grant #P200A980307.

References

- [1] A. Kiely and M. Klimesh. The ICER progressive wavelet image compressor. *The Interplanetary Network Progress Report 42-155, July-September 2003, Jet Propulsion Laboratory*, pages 1-46, Nov 2003.
- [2] A.R. Calderbank, I. Daubechies, W. Sweldens, and B.L. Yeo. Lossless image compression using integer to integer wavelet transforms. In *Proceedings of the 1997 International Conference on Image Processing (ICIP '97)*, pages 596-599. IEEE Computer Society, 1997.
- [3] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Addison-Wesley, Boston, MA, 1992.
- [4] M. Swanson and A. Tewfik. A binary wavelet decomposition of binary images. *IEEE Transactions on Information Processing*, 3(12):1637-1650, Dec 1996.
- [5] <http://www.cog.brown.edu/~tarr/stimuli.html>.
- [6] ftp://ftp.vislist.com/IMAGERY/MED_3D_SLICES.
- [7] <ftp://ftp.funet.fi/pub/graphics/misc/test-images/>.
- [8] <http://bias.csr.unibo.it/fvc2000/download.asp>.
- [9] <http://sipi.usc.edu/services/database/>.
- [10] <http://links.uwaterloo.ca/bragzone.base.html>.
- [11] A. A. Mirin et al. Very high resolution simulation of compressible turbulence on the IBM-SP system. Technical Report UCRL-JC-134237, Lawrence Livermore National Laboratory, 1999.