OPTIMIZATION OF THE SW ALGORITHM FOR HIGH-DIMENSIONAL COMPRESSION

Krasimir Kolarov, William Lynch

Interval Research Corporation, 1801-C Page Mill Road, Palo Alto, CA 94304, USA
(415)842-6045 phone, E-mail contact: kolarov@interval.com

ABSTRACT

This paper describes an algorithm and a software package SW (Spherical Wavelets) that implements a method for compression of scalar functions defined on 3D objects. This method combines discrete second generation wavelet transforms with an extension of the embedded zerotree coding method. We present some results on optimizing the performance of the SW algorithm via the use of arithmetic coding, different scaling and norms of the wavelet coefficients. We describe an extension of the SW algorithm using different prediction schemes in the zerotree mechanism. The combined use of those techniques leads to a significant improvement of the compression performance of SW.

PART I INTRODUCTION

We have introduced recently [1] a new technique for compression of scalar functions defined on 2-manifolds of arbitrary topology. This technique resulted in a software package for compression SW (for Spherical Wavelets) which combines discrete wavelet transforms with zerotree compression, building on ideas from three previous developments: the lifting scheme, spherical wavelets, and embedded zerotree coding methods. The lifting scheme and the spherical wavelets are techniques introduced recently [2, 3] for enabling wavelet construction for more general cases than the typical 1D or 2D planar spaces. In that, the wavelet coefficients are generated through a simple linear prediction and update scheme that can be applied to space of any dimension. This is a multi-resolution scheme where in the coarsest level the object is represented by a simple base complex (e.g. a triangular mesh in 3D). All the cells of this complex are subdivided to generate the next level of approximation and the corresponding wavelet coefficients are computed for the newly generated vertices. This process is continued until appropriate coverage of the data set is achieved.

The zerotree algorithm was introduced [4, 5] for effective and fast embedded progressive compression of images. In our context that algorithm processes the wavelet coefficients generated from the transform analysis part based on significance with respect to given threshold. The coefficients are arranged in a tree structure whereby the main premise of the method is that if certain coefficient is insignificant with respect to the threshold, all coefficients below that one in the tree are also insignificant. That allows for the tree to be pruned and a smaller set of coefficients are written out representing the data. We used few basic ideas from the zerotree implementation in [5] to perform the entropy coding. The zerotrees as implemented in classical image coding are taking strong advantage of the fact that the objects being processed are flat rectangular images. Thus we needed to redesign and re implement the data and control structures of the algorithm for general 2D manifolds (sphere in particular).

In what follows we will describe some experiments in parameter analysis and optimization of the SW algorithm.

PART II THE COMPRESSION PROCESS IN SW

2.1 Experimental Setup

To recall from [1] we are using two main data sets:

- For the Earth example we compressed the topographic function that is the elevation (with respect to sea level) of the Earth. This function is approximated from the ETOPO10 data set which samples the Earth from satellites every 10 arc minutes (1.5 million points approximately 11 miles apart). For the base complex, in the terminology of [1] and [2], we chose the icosahedron and subdivided it eight times yielding 655362 vertices and 1310720 triangles. The vertices of this complex were then radially projected onto a concentric geoid (vertices average about a 22 arc minute separation - about 25 miles). The topographic elevation at each vertex was then determined by interpolation of the ETOPO10 data set (2:1 reduction in points).

- For comparison with existing data compression techniques we used SW to compress the Lena image (the function being compressed is the gray-scale level at each pixel). We specify a base complex consisting of two triangles, thereby forming a square. This was then subdivided nine times yielding a complex with 513^2 vertices and $2 * 512^2$ triangles. The function value at each of the 513^2 vertices was determined by interpolation from a 512^2 gray-scale Lena image.

For both data sets Table 1 presents the results using linear lifting and various compression ratios. Scaling was chosen appropriate to the L_2 norm.

2.2 Compression Evaluation

The first column of the table describes how many bit planes are read during decompression by the algorithm. The term "compression ratio" tends to be misleading and in the context of SW is even more so. For example in the case of the Earth data, we never actually work directly with the original satellite data. Because of the limitations of the hardware, we were not able to process the full ETOPO10 data set. Even if we could generate enough coefficients to cover all data points, because of the way the coefficients are distributed along the surface of the sphere (actually the subdivided icosahedron), their values are always interpolated. Thus we evaluate compression by looking at the amount of information that is read during decompression in relation to the amount of information that gets written out by the compression module.

One of the input parameters to SW is the number of bitplanes used for representing each wavelet coefficient. For Table 1 below we have used 10 bit planes. During compression the wavelet coefficients generated by the lifting are scaled and then further processed by the zerotree construction. The resulting output stream consists of coefficients (in order of magnitude) from the most significant to the least significant bitplanes. This output stream includes ALL the bitplanes for the coefficients as truncated by the zerotree hypothesis.

At decompression we can decide (based on resources - time and space) to read in all or just a portion of the output bit stream written during compression. Thus the first column in the Table 1 shows how many bitplanes were read back starting from the most significant one. This ability reflects the "progressive coding" character of the code.

The second column reflects the true measure for compression performance which together with the PSNR described below can be used for comparison with other algorithms. In our case the notion of bpp (bits per pixel) correspond to bits per (wavelet) coefficient, since those are the ones that represent the data structure. Thus for example in the first row when 10 bit planes were read in during decompression, the actual size of the file was 369189 bits for the Lena image. Those correspond to 513^2 = 263169 wavelet coefficients, yielding a rate of 1.4029 bpp.

Although as we mentioned compression ratio is not a good criteria in general, it is a widely used characteristic for comparison of algorithms. The correct way for us to calculate the compression ratio is with respect to the amount of data needed to write out directly all the wavelet coefficients. Every coefficient takes 8 bits for complete representation. Thus in the example of the first row of the table for the Lena image, the 263169 coefficients will need 2105352 bits for complete representation. Using 10 bit planes per coefficient our algorithm reads (and first writes) 369189 as outlined. Thus the true compression ratio is 6:1 for those parameters. As we will see later we can achieve a 1:1 ratio by taking more bitplanes per coefficient (in particular for this example 15 bit planes achieve that ratio).

	LENA (9 levels, 263169 coeff.)				EARTH (8 levels, 655362 coeff.)			
bitplanes	b/vertex	PSNR	Compress	Arithmetic	b/vertex	PSNR	Compress	Arithmetic
10	1.4029	38.83	6:1	1.3597	0.5665	41.96	14:1	0.5378
9	0.5428	34.25	15:1	0.5220	0.1968	37.51	41:1	0.1864
8	0.2533	30.39	31:1	0.2435	0.0802	34.11	100:1	0.0748
7	0.1177	27.11	68:1	0.1136	0.0335	31.12	240:1	0.0310
6	0.0505	24.18	160:1	0.0489	0.0142	28.52	565:1	0.0131
5	0.0211	21.64	382:1	0.0205	0.0059	26.06	1356:1	0.0055
4	0.0092	19.34	866:1	0.0091	0.0027	23.98	2994:1	0.0024
3	0.0032	16.71	2474:1	0.0032	0.0014	22.03	5804:1	0.0013
2	0.0014	15.09	5751:1	0.0014	0.0008	19.85	9927:1	0.0008
1	0.0007	13.97	10961:1	0.0007	0.0007	18.65	11579:1	0.0007

Table 1 PSNR data for the Earth (8 levels of subdivision, 1 vertex every 25 miles) and Lena (9 levels of subdivision) examples for different compression ratios.

During decompression the wavelets are descaled by the inverse of the scaling factor used during compression.

2.3 Scaling

As explained in [1] we can optimize for the different norms by scaling the coefficients after the transform analysis and before the zerotree compression for a given threshold. Multiplication by unity optimizes for the L_∞ norm. To optimize for the L_2 norm the wavelet coefficients should be scaled by 2^{-j} where j is the subdivision level at which the coefficient is attached. This basically corresponds to shifting the bits in the coefficients' representation one to the left. To optimize for the L_1 norm the wavelet coefficients should be scaled by 4^{-j}. This corresponds to two bits shift in the coefficients. For the general L_p norm the scaling factor is $4^{\frac{-j}{p}}$. The contrast sensitivity curve for the human visual system is best approximated by using the L_1 norm and its associated scaling [6].

We experimented with L_2 , L_1 and L_∞ scaling of the coefficients. The results with L_2 scaling are outlined in Table 1. The comparative results for the Lena image with L_1 and L_∞ norm are presented in Table 2.

LENA	L1 norm (shift 2 positions left)			L infinity norm (no scaling)		
bit planes	b/vertex	PSNR (dB)	Compression	b/vertex	PSNR (dB)	Compression
10	0.0384	23.18	209:1	9.7221	68.72	1:1.2
9	0.0188	21.37	424:1	8.1771	60.71	1:1.02
8	0.0108	19.68	740:1	6.584	52.83	1.2:1
7	0.0058	17.85	1377:1	4.9062	45.65	1.6:1
6	0.0035	16.76	2319:1	3.1478	39.81	2.5:1
5	0.002	15.63	3972:1	1.5931	33.98	5:1
4	0.0014	15.06	5832:1	0.6906	28.4	12:1
3	0.0009	14.17	9075:1	0.2728	23.56	29:1
2	0.0008	14.01	10580:1	0.0815	18.4	98:1
1	0.0007	13.41	12240:1	0.0129	14.35	620:1

Table 2 PSNR data for the Lena (9 levels of subdivision, 263169 coefficients) example for different scaling and different compression ratios.

From those tables we can observe that the performance curves for the L_1 and the L_2 scaling are actually the same. All points from the L_1 portion of Table 2 lie on the curve for L_1 but are much earlier on the curve than the corresponding bitplanes points from L_2 norm. We can obtain L_1 points further along the curve by using more bitplanes for the coefficients (e.g. 15 bit planes yield a point with 32.36 dB for 0.45 b/vertex). Since we are shifting two positions left for L_1 vs. one position left for L_2 it is natural to expect that we will get a smaller output stream, that will bring smaller b/vertex rate and correspondingly smaller PSNR for the same number of bitplanes read. That the curves are the same however merits further investigation.

The other main conclusion from the tables above is that any scaling (L_1 or L_2)do much better than no scaling at all (i.e. L_∞ norm). In fact using scaling gives us a 6 dB improvement over the whole range of the performance curve. For the reminder of the report we will consider L_2 scaling of the coefficients.

PART III PARAMETER OPTIMIZATION IN SW

3.1 Available parameters

In this Section we will describe some of the experiments we performed with varying the different parameters in the algorithm in order to achieve the best compression performance. In particular Section 3.2 considers the influence of the number of levels of subdivision in the algorithm, while Section 3.3 concentrates on the role of the number of bitplanes used to represent the coefficients. Section 3.4 compares the use of different prediction schemes.

3.2 Levels of subdivision of the base manifold

Earlier we mentioned that we run the experiments with the Earth example with maximum 8 levels of subdivision because of hardware limitations. We would like to point out that there is a

significant advantage of using as many subdivision levels as possible in order to cover all the data points in the original database. Extensive experiments show that the tables adding one more level of subdivision increases the PSNR with more than 5 dB for both the Lena image and the Earth data. This justifies our goal of getting as much subdivision as possible. Naturally we don't need more coefficients than the number of data points.

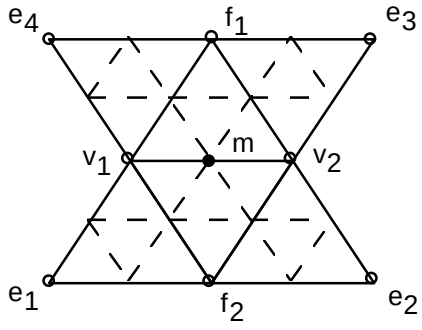
3.3 Number of bitplanes for the wavelet coefficients

In the examples so far we have used 10 bit planes to represent the wavelet coefficients. As we can see from the data for L_1 normalization in Table 2 in some cases we might need more bitplanes in order to obtain higher PSNR and better visual perception quality. Naturally for images like Lena the maximal bits per pixel (coefficient) rate that is needed is 8 bpp, since that is what it takes tot represent all coefficients explicitly. In fact in order to obtain a 1:1 basis for visual comparison of the images we need to find this maximal number of bitplanes needed.

For the example of the Lena image above with 9 levels of subdivision we obtained a 1:1 compression ratio (as explained in Section 2.2) for 15 biplane representation of the coefficients. The resulting PSNR is 66.56 dB for 8.15 bpp. If we calculate the performance curve for reading in at decompression 15, 14, 13, ..., 2, 1 bitplanes of the 15 bitplanes written out at compression, we can see that this is exactly the same curve as the one for the points in Table 1 with the new points beyond 10 bitplanes lying further to the right on the curve. We have used a similar 1:1 picture as the base image for comparison with the compressed ones in Figure 6.

3.4 Linear vs. Butterfly prediction results

All the examples so far have used the linear prediction scheme outlined in [1]. There the new wavelet coefficients generated at the midpoints of any given edge are calculated as the mean of the two coefficients associated with the vertices of the edge. The other scheme described in [1] is the Butterfly scheme where in Figure 1 the new coefficient is a linear combination of eight neighboring vertices from the previous level, i.e. we use a larger stencil for prediction of a new coefficients. We performed some simulations to evaluate the performance of the algorithm in that case.



$$m \approx \frac{1}{2}(v_1 + v_2) + \frac{1}{8}(f_1 + f_2) - \frac{1}{16}(e_1 + e_2 + e_3 + e_4)$$

Figure 1 The Butterfly scheme for prediction of values for the wavelet coefficients.

In order to create a 2D manifold from the flat square image for Lena, we glued two copies of the image back to back. The edges on the boundary of the image were associated with both the upper and the lower copies. Unfortunately the construction broke the symmetry required for the

Butterfly scheme and we were not able to run experiments for that case. This fact does not at all affect the compression results for the Lena described so far, since the top level (original) vertices in either case are directly written out by SW and are not compressed or decompressed.

Even more importantly, to test the different prediction schemes we actually needed a "true'" 3 dimensional structure which is exhibited by the Earth example. The results for the compression curve for 8 levels of subdivision with Butterfly prediction are compared with those with Linear prediction in Figure 2. As we can see from the figure as well as from Table 4 in the next Section, Butterfly brings only about 4 % improvement over the linear case. This is probably due to the fact that the geometry of the mesh approximation of the Earth sphere is quite regular. We expect to achieve significant improvement in performance when using larger stencils for high-dimensional objects with sharp edges and concavities.

PART IV ENTROPY CODING IN SW

4.1 Description

In this Section we will describe how the addition of entropy coder and modifications of the coding scheme affect the performance of our algorithm. Section 4.2 introduces an arithmetic coder that we implemented to supplement SW. Section 4.3 describes an alternative way of building the G-tree in the zerotree implementation that allows us to achieve a significant improvement of more than 12% compared to the results described in [1].

4.2 Arithmetic coding of the output binary stream

The result from applying the wavelet transform, scaling of the wavelet coefficients and the zerotree compression is an output bit stream. We can try to further enhance the performance curve by applying Huffman coding or arithmetic coding to this output stream.

One simple experiment that we performed was to run LZW compress on the output stream. The result was that there was virtually no reduction in the size of the file. That lead us to believe that the previous steps in the compression algorithm already have exploited all possible symmetries and runs in the data. We also experimented with trying to reorder the output of the sign and significance bits from the zerotree algorithm. This reordering brought up some improvement in the compression curve which will be exploited further in the future.

In the description of the Said and Pearlman zerotree implementation [5] an arithmetic coder was used for further run-length encoding of the bit stream. We decided to implement an arithmetic coder and apply it to the compressed output of SW. The results for few of the settings are summarized in columns 5 and 9 of Table 1. The arithmetic coder that we implemented is the well-known Witten coder described in [7].

The actual code implementation generates the arithmetic coded output stream during decompression in parallel with reading in the compressed data. Naturally the PSNR does not get affected by the arithmetic coding, just the size of the bit stream is smaller. Thus Table 1 reports the b/vertex results for binary and arithmetic coding corresponding to the same PSNR. Graphically Figure 3 compares the performance curves for the Lena image with 9 levels of subdivision with and without the use of arithmetic coder.

As we can see from the table, arithmetic coder brings on the average 3.3 % improvement over the straight binary coding. This result is similar to the one reported in [5]. The conclusion is that in general the added expense of performing arithmetic coding outweighs the benefit of using it in SW.

4.3 Zerotree Coding

Our implementation of the zerotree compression using G-trees is described in detail in [1]. As pointed out in that report, the main structure of the algorithm bears similarity with the SPIHT (Algorithm II) from [5]. However we had to introduce new data structures (the G tree) and control flow to account for the more complicated topology of the spaces we are processing. Here we will briefly mention some results for varying initialization and branching phase of the algorithm.

To recall from [1] we define G IT as a list of triangles in the G-tree, GIV and GSV are lists of vertices in the G-tree. The first part of the algorithm is:

1. **Initialization**:

   Set $n = N = sig(D(R))$ output N

   Initialize GIT to the triangles in the base complex (as type A triangles)

   Initialize GIV to the vertices in the base complex

   Initialize GSV to empty

Here *D(R)* is the set of descendants of the root of the tree *R*.

As we mentioned earlier, in the actual implementation of the algorithm the top level coefficients at the vertices (4 for the square defining the Lena image and 20 for the icosahedron defining the Earth approximation) are not processed at all from SW. They are basically written out directly in the output stream. This creates a choice for the initialization of the data structures.

One option is to initialize GIT with the top level triangular faces (2 for the square and 20 for the icosahedron) and initialize GIV to empty (since we are not processing the top level vertices). Using this choice the fan out on the next level per vertex is maximum 4 and we can achieve a good depth in the tree. In that case the significance of a vertex is the value that is calculated for it. The significance of a triangle is based on the significance of the triangles into which it is subdivided as well as the significance of its vertices. This is the implementation described in [1].

Alternative way to implement the algorithm is by initializing GIV with the midpoints of the edges of the base triangles in GIT (5 in the case of the square and 30 for the icosahedron). We can not use the original top level vertices because they are not processed. This structures requires calculating significance of all triangles of Type A by looking at their children (triangles) in the tree and their vertices. The significance of all triangles of type B (already encountered) is calculated using the significance of the descendants except itself and its children. The fan out of this approach is 8 per node and the G tree is generally not as deep as in the previous case.

The numerical results of the simulations with both structures are shown in Table 3. As we can see the first method has very similar performance curve to the second one. The improvement for the second one is about 4% on the average for the Lena image and 1.75% on the average for the Earth.

However if we add arithmetic coding to the wider G-tree generation we achieve a significant improvement. The compression curve becomes on the average 10% better than the resolution [1]. Figure 4 compares the results from [1] with those using wider G-tree (High Fan-out) and arithmetic coding for the Earth example (8 levels of subdivision, Linear basis). Similar comparison for the Lena image with 9 levels of subdivision is illustrated in Figure 5. As we can see from that figure, the improvement in performance is more than 1 dB. In addition the numbers now very competitive with the ones reported in [5] for the flat image case.

	LENA (9 levels, 263169 coeff.)			EARTH (8 levels, 655362 coeff.)		
bit planes	b/vertex	PSNR (dB)	Compression	b/vertex	PSNR (dB)	Compression
10	1.2572	38.83	6.4:1	0.5325	41.96	15:1
9	0.506	34.25	16:1	0.1893	37.51	42:1
8	0.2399	30.39	33:1	0.078	34.11	103:1
7	0.1132	27.11	71:1	0.0327	31.12	245:1
6	0.0492	24.18	162:1	0.0139	28.52	574:1
5	0.0207	21.64	387:1	0.0057	26.06	1395:1
4	0.009	19.34	889:1	0.0026	23.98	3079:1
3	0.0031	16.71	2543:1	0.0014	22.03	5904:1
2	0.0014	15.09	5800:1	0.0008	19.85	9620:1
1	0.0007	13.97	11023:1	0.0007	18.65	11156:1

Table 3. Wider G-tree results for the Earth (8 levels of subdivision, Linear basis) and Lena (9 levels of subdivision, Linear basis) examples for different compression ratios.

PART V ANALYSIS AND CONCLUSION

As we pointed out in [1] the results that we obtained for the Earth data visually look exactly as what we would expect for good compression given the resolution that we are achieving. Since that is the first method that allows to do that type of compression for anything other than flat images (or sequences of such) it is difficult to judge the PSNR numbers we are achieving.

On the other side the PSNR numbers for Lena are quite comparable with the ones reported for the embedded zerotree in [5] (there for compression 16:1 the PSNR is 33.79 dB). This is despite the fact that the method and the algorithm in [5] are custom crafted to take advantage of the flat spatial structure images. We have illustrated the visual results from the simulations for linear basis, 9 levels of subdivision, with arithmetic coding and wider G-tree for the Lena image in Figure 6. It also shows the Earth example with 8 levels of subdivision, wider G-tree and Butterfly base.

Using the analysis in this paper we can improve the performance described in [1]. The Butterfly scheme allows a 4% improvement over the Linear one. For a span of 4 children per vertex the arithmetic coding improves the performance by 4% on the average. For the span of 8 children the arithmetic coder brings about 10% improvement. We also showed that the best results are achieved as many subdivision levels as necessary to cover the original data. More bitplanes allows us to get compression ratios close to 1:1 and obtain base pictures for comparison. Finally it is beneficial to use scaling and L_2 seems to be most appropriate norm for good compression curve. As seen in Figure 1 using a combination of these techniques brings about 12% improvement in performance with respect to the results in [1].

REFERENCES

[1] K. Kolarov and W. Lynch Compression of functions defined on surfaces of 3D objects, *Proceedings of the Data Compression Conference*, Snowbird, 1997.

[2] P. Schröder and W. Sweldens. Spherical wavelets: Efficiently representing functions on the sphere, *Computer Graphics Proceedings, (SIGGRAPH 95)*, pages 161--172, 1995.

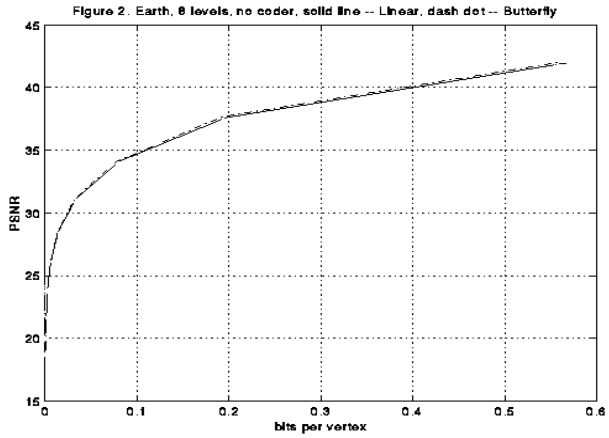
[3] W. Sweldens. The lifting scheme: A custom-design construction of biorthogonal wavelets, Technical Report 1994:7, Industrial Mathematics Initiative, Department of Mathematics, University of South Carolina, 1994.

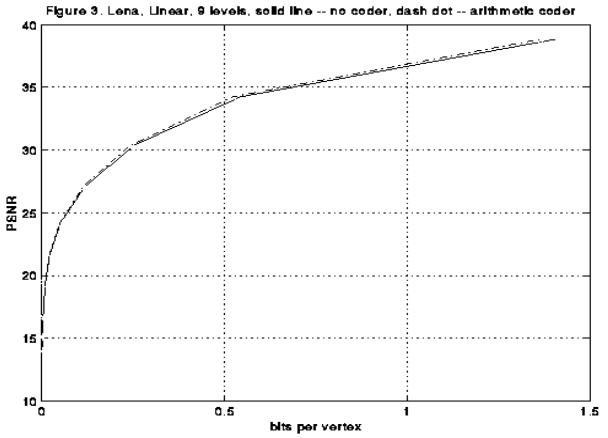
[4] J.M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients, 41(12):3445--3462, 1993, IEEE Trans. Signal Process.

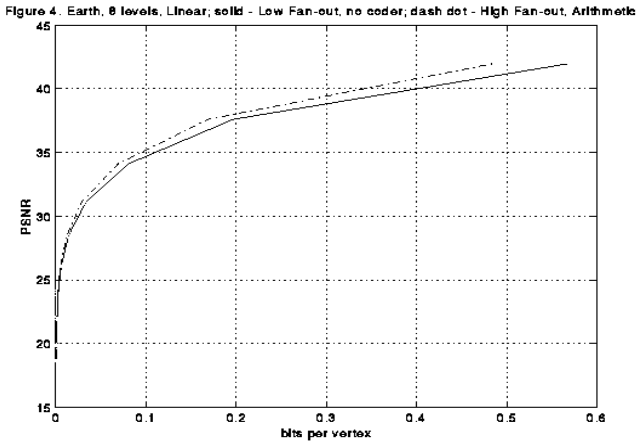
[5] A. Said and W. A. Pearlman. A new fast and efficient image codec based on set partitioning in hierarchical trees, Submitted to the IEEE Transactions on Circuits and Systems for Video Technology.

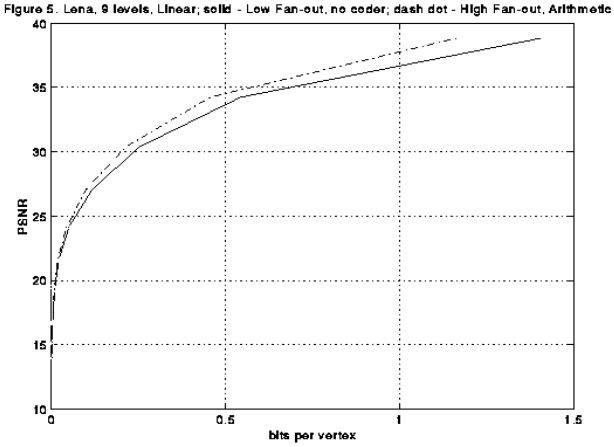
[6] R. A. DeVore, B. Jawerth, and B. J. Lucier, "Image compression through wavelet transform coding", *IEEE Trans. Inform. Theory*, vol. 38, pp. 719-746, March 1992

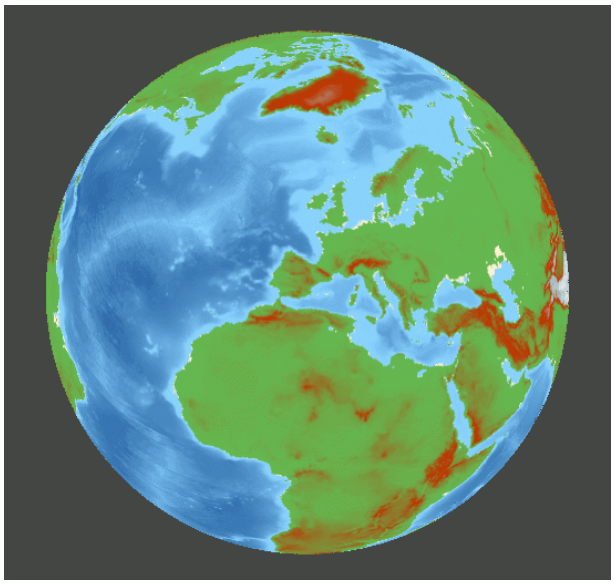
[7] I. Witten, R. Neal and J. Cleary, "Arithmetic coding for data compression", In *Communications of the ACM*, v. 30, n. 6, pages 520-540, June 1987.



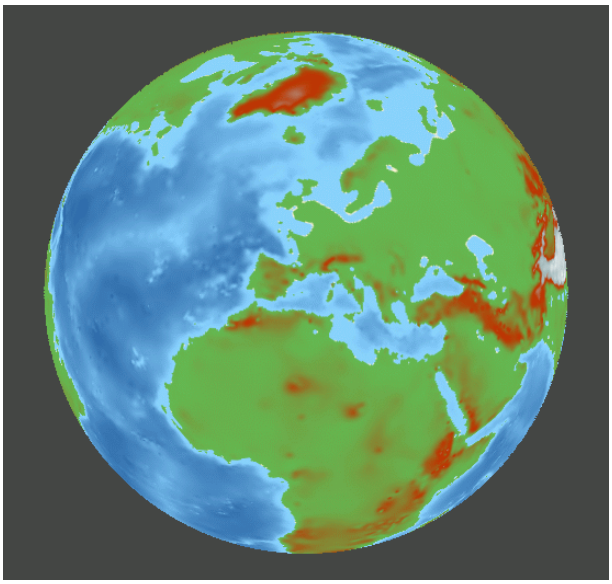








19 bit planes, 1:1 compression
PSNR = 89.92 at 7.97 bpv



9 bit planes, 48:1 compression
PSNR = 37.68 at 0.17 bpv

17 bit planes read,
PSNR = 78.58 at 8.10 bpp

8 bit planes read,
PSNR = 30.39 at 0.21 bpp

Figure 6 Earth data (Butterfly base, wide G-tree, 8 levels of subdivision, arithmetic coder) and Lena data (Linear base, wide G-tree, 9 levels of subdivision, arithmetic coder).