GVU Technical Report. May 2003. To appear in Eurographics 2003.

Out-of-core compression and decompression of large *n*-dimensional scalar fields

Lawrence Ibarria*, Peter Lindstrom†, Jarek Rossignac* and Andrzej Szymczak*

*GVU Center, College of Computing, Georgia Institute of Technology, Atlanta, USA
†Lawrence Livermore National Laboratory, Livermore, USA.

Abstract

We present a simple method for compressing very large and regularly sampled scalar fields. Our method is particularly attractive when the entire data set does not fit in memory and when the sampling rate is high relative to the feature size of the scalar field in all dimensions. Although we report results for $\mathbb{R}^3$ and $\mathbb{R}^4$ data sets, the proposed approach may be applied to higher dimensions. The method is based on the new Lorenzo predictor, introduced here, which estimates the value of the scalar field at each sample from the values at processed neighbors. The predicted values are exact when the n-dimensional scalar field is an implicit polynomial of degree $n-1$. Surprisingly, when the residuals (differences between the actual and predicted values) are encoded using arithmetic coding, the proposed method often outperforms wavelet compression in an L_∞ sense. The proposed approach may be used both for lossy and lossless compression and is well suited for out-of-core compression and decompression, because a trivial implementation, which sweeps through the data set reading it once, requires maintaining only a small buffer in core memory, whose size barely exceeds a single $(n-1)$-dimensional slice of the data.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Compression, scalar fields, out-of-core.

1. Introduction

Numerous engineering, biomedical, and other scientific applications produce extremely large data sets through numeric simulations or physical data acquisition. In a large proportion of the cases, the data represents one or more scalar fields sampled over regular grids in dimension three, four, or higher. For example a typical 3D simulation produces values on a regular grid of $2,048^3$ samples [6]. In 4D a typical combustion simulation generated using a High-Performance Parallel Processing Cluster may include 1,000 time slices, each representing a regular sampling of a cube at a resolution of 512^3 [1]. Another example may be fluid dynamics data, used in our tests (see Fig. 1). At each space-time sample, the values of several scalar and vector fields are produced. Storing the results of this simulation and transmitting them to remote visualization clients is expensive. A variety of data compression techniques have been proposed to reduce the storage and transmission cost [4].

We focus here on the loss-less, single resolution (i.e. non progressive) compression. Instead of a hierarchical method, which transmits a sub-sampled (and possibly smoothened) model first and then estimates the missing values through interpolation, we transmit the values in order, using a new predictor to extrapolate the next value from the previous ones. The residuals (differences between the actual and predicted values) may be encoded with fewer bits and, if desired, further compressed using arithmetic coding.

We have extended a simple two-dimensional parallelogram predictor [5] to higher dimensions and have named it the Lorenzo predictor. It estimates the scalar value of a sample on the corner of an *n*-dimensional cube from the scalar values of the others 2^n-1 corners. Although the formula for the predictor is very simple, its predictive power is significant for higher-dimensional data. For example, in $\mathbb{R}^4$, the Lorenzo predictor can recover exactly any scalar field that corresponds to an implicit cubic polynomial. In some situations, the proposed method outperforms wavelet compression in an L_∞ sense, when the residuals are encoded using arithmetic compression. Furthermore, because, during compression and decompression, we only need to access the immediate neighbors, the proposed approach is particularly well suited for out-of-core compression and decompression

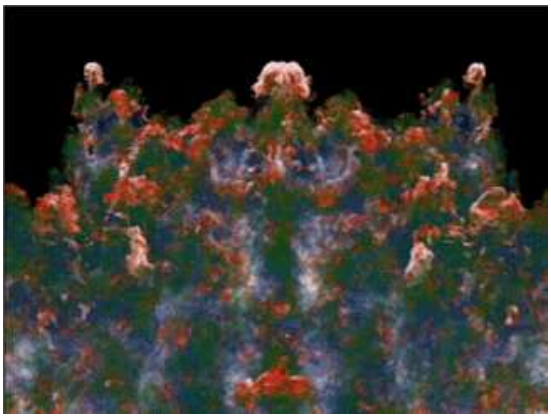

Figure 1: *A 4D data set from a simulation of two fluids interacting.*

where the total size of the data set significantly exceeds what can be stored in core memory.

2. Prior Art

Several compression techniques have been proposed for lower dimensional gridded data. These include image and video compression techniques, as offered by the MPEG-4 standard [18] and various volume compression approaches [11, 12].

A variety of methods to compress 4D volumes have been proposed in recent years. These include wavelets [10], discrete cosine transform (DCT) [3] and run length encoding (RLE) [2]. The wavelet approach uses an interpolating predictor, which, according to informal experiments, produces 50% smaller residuals than an extrapolating predictor. On the other hand, the wavelet's hierarchical approach requires more space and processing power than our extrapolating predictor. When local temporary storage is an issue, wavelet approaches may break the data set in smaller chunks and compress each one independently. The proposed approach does not require such splitting.

Fowler and Yagel [12] propose a similar approach to ours. They also use previously decoded samples to predict a new value in 3D volumes. They use the three nearest visited neighbors, and compute optimal coefficients for their predictor. In contrast, we use seven neighbors. Ma et al [13] compute an octree for each frame, and encode the differences between octrees. They also compare uniform and non-uniform (or adaptive) quantization of the data before compression.

Others quantize the corrections or the wavelets coefficients, e.g. Bajaj et al. [11]. On the contrary, we quantize the data set and then perform lossless encoding. As a result, the maximum error produced by our approach is given by the quantization error.

Out-of-core methods for simplifying [14] and compressing [15, 17] 3D polygonal meshes have recently been proposed. Out-of-core methods for compressing 3D volumetric data sets have also been proposed [16]. Chiueh et al's approach segments the volume into different chunks, transforms each chunk separately using a Fourier transform, and encodes the transformed result.

3. The Predictor

When applied to a sample v, the Lorenzo predictor estimates the scalar value $F(v)$ at v from its immediate neighbors that have already been processed. Assume that the data is organized as a regular grid of samples. Both the compressor and decompressor visit the data in scanline order. For simplicity of notation, we use the local coordinate system where sample v has coordinates $\{1\}^n = (1,1,\ldots,1)$ and its previously visited neighbors are those samples with coordinates in $\mathbb{Z}_2^n = \{0,1\}^n$. The value of the scalar field $F(v)$ is estimated from the field values at the other previously recovered vertices $U = \mathbb{Z}_2^n - \{v\}$ of the n-dimensional unit cube using the following formula:

$$E(v) = \sum_{u \in U} (-1)^{c_0(u)+1} F(u) \qquad (1)$$

where $E(v)$ is the prediction of F at v and $c_0(u)$ denotes the number of coordinates of u that equal zero. Note that $c_0(u)$ may also be expressed as $c_0(u) = n - c_1(u) = n - u \cdot v$, where n is the dimension, $c_1(u)$ is the number of coordinates in u that equal one, and $u \cdot v$ is the dot product of u and v.

Note that, as shown in Fig. 2, in this formulation, the immediate neighbors of the predicted vertex v have weight $+1$. Second degree neighbors (i.e., those which can be reached from v by traversing two edges of the cube) have weight -1, third degree neighbors have weight $+1$, and so on.

4. Prediction for polynomials

The estimated values computed by the Lorenzo predictor in n dimensions are exact for all scalar functions that are polynomials of degree $n-1$. As a proof, assume that P is a polynomial of degree m in n variables, with $m < n$, and consider the following theorem and its corollary:

Theorem 1 For a given monomial M in n variables of degree $m < n$, the sum of signed values $(-1)^{c_1(u)} M(u)$ over all the vertices u of the unit cube is zero.

More formally, let $u = (x_1, \ldots, x_n)$. A monomial $M(u)$ has the form: $x_1^{p_1} \cdots x_{k-1}^{p_{k-1}} x_k^{p_k} x_{k+1}^{p_{k+1}} \cdots x_n^{p_n}$, with $\forall i \; p_i \geq 0$ and $\sum_i p_i = m$. The theorem states that $\sum_{u \in \mathbb{Z}_2^n} (-1)^{c_1(u)} M(u) = 0$.

Proof There are n variables, but the sum $\sum_i p_i = m$ of the powers of the variables listed in M is less than n. Therefore at least one variable is not listed in M. Assume without loss of generality that the k^{th} variable is not listed. Consequently, the value of M is independent of that variable and thus $M(x_1, \ldots, x_{k-1}, 0, x_{k+1}, \ldots, x_n) =$

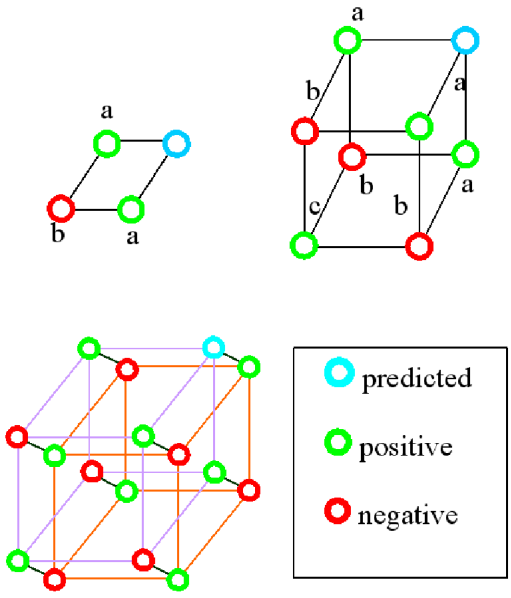

Figure 2: *In the 2D case (top left), the new value is predicted from its neighbors using the parallelogram rule (add the scalar field values at the two 'a' vertices and subtract the value at the 'b' vertex). In the 3D case, we add the values of the 'a' corners, subtract the values of 'b' corners, and add the values of the 'c' corner. In the 4D case, we add the values at the first and third degree neighbors and subtract the sum of the values at the second and fourth degree neighbors.*

$M(x_1,\ldots,x_{k-1},1,x_{k+1},\ldots,x_n)$. Note that $(-1)^{x_1+\cdots+0+\cdots+x_n} = -(-1)^{x_1+\cdots+1+\cdots+x_n}$. Therefore, the vertices of the cube can be paired so that the values of $(-1)^{c_1(u)}M(u)$ on the two vertices of any pair are either both zero or have the same magnitude but opposite signs. Thus, the sum of the signed values is zero. □

This result may be easily extended to polynomials as follows.

Corollary 1 For a given polynomial P in n variables of degree $m < n$, the sum of the signed values $(-1)^{c_1(u)}P(u)$ over all the vertices u of the unit cube is zero.

Proof $P = \sum_i M_i$ is the sum of monomials of degree $m < n$. We can permute the two summations:

$$\sum_{u\in\mathbb{Z}_2^n}(-1)^{c_1(u)}P(u) = \sum_{u\in\mathbb{Z}_2^n}(-1)^{c_1(u)}\sum_i M_i(u)$$
$$= \sum_i\sum_{u\in\mathbb{Z}_2^n}(-1)^{c_1(u)}M_i(u) = \sum_i 0 = 0$$

which proves the corollary. □

The corollary implies that

$$(-1)^n P(1,\ldots,1) = -\sum_{u\in U}(-1)^{c_1(u)}P(u)$$

and hence

$$P(v) = (-1)^{n+1}\sum_{u\in U}(-1)^{c_1(u)}P(u) = \sum_{u\in U}(-1)^{c_0(u)+1}P(u)$$

As a consequence, in two dimensions the Lorenzo predictor is a linear predictor, and can exactly reconstruct portions of the scalar field that behave as a linear function

$$F(x,y) = ax+by+c$$

In $\mathbb{R}^3$, the same simple Lorenzo predictor can reconstruct quadratic functions:

$$F(x,y,z) =$$
$$ax^2+by^2+cz^2+dxy+exz+fyz+gx+hy+jz+k$$

In $\mathbb{R}^4$, the predictor extends its reconstruction power to all cubic polynomials, which are linear combinations of 35 possible monomials of degree of 3 or less in 4 variables. Even though the 15 values at neighboring grid points that the Lorenzo predictor uses do not provide enough information to compute all 35 coefficients of the polynomial, they uniquely determine its value at the corner v.

The Lorenzo predictor is of highest possible order among all predictors that estimate the value of a scalar field at one corner of a cube from the values at the other corners. In other words, it is of optimal order for this setting, and no other predictor can correctly estimate all polynomials of degree n or higher.

As a justification, consider the monomial $x_1x_2\cdots x_n$ (the product of all coordinates). The product is zero on all vertices of the unit cube except for one. So, a predictor would not be able to differentiate this monomial from the zero polynomial. Hence, the values of the scalar field at the 2^n-1 corners of an n-dimensional cube are not sufficient to recover the value of an n^{th} degree polynomial at the 2^{nth} corner.

Note that simpler predictors exist that correctly predict all polynomials of degree $m < n$. For example, one may use the n samples that precede v on a scanline. However, such lower-dimensional anisotropic predictors are much less effective since they fail to exploit data coherence in all dimensions. The Lorenzo predictor is the simplest isotropic predictor that can recover correctly all polynomials of degree less than n.

5. Scanline compression algorithm

Consider a 4D scalar data set organized in an array $F[xmax,ymax,zmax,tmax]$. Pseudocode for the compression algorithm is presented below:

```
Lorenzo(d, x_1, ..., x_n)
    if all x_1, ..., x_n differ from -1
        E := LorenzoPredictor(d, x_1, ..., x_n)
        Encode(F[x_1, ..., x_n] - E)
    else
        Lorenzo(d - 1, x_1, ..., x_{k-1}, 0, x_{k+1}, ..., x_n)
        for i = 1 to dmax do
            Lorenzo(d, x_1, ..., x_{k-1}, i, x_{k+1}, ..., x_n)
```

The variable d indicates the dimension of the predictor, $x_1,\ldots,x_n$ are the coordinates of a sample in the data, *dmax* is the number of samples in the d^{th} dimension, and k denotes the greatest index between 1 and n where x_k equals -1. The function *LorenzoPredictor* computes the Lorenzo predictor of dimension d (the first parameter) at the coordinates $x_1,\ldots,x_n$. The starting call to compress a 4D data set would be:

$$Lorenzo(4,-1,\ldots,-1)$$

For example consider the 2D case of a 3×3 matrix H, with points from $(0,0)$ to $(2,2)$. The trace of the compression program on the integer lattice would be:

call	*encoded value*
$Lorenzo(2,-1,-1)$	
$Lorenzo(1,-1,0)$	
$Lorenzo(0,0,0)$	$H[0,0]$
$Lorenzo(1,1,0)$	$H[1,0]-H[0,0]$
$Lorenzo(1,2,0)$	$H[2,0]-H[1,0]$
$Lorenzo(2,-1,1)$	
$Lorenzo(1,0,1)$	$H[0,1]-H[0,0]$
$Lorenzo(2,1,1)$	$H[1,1]-(H[1,0]+H[0,1]-H[0,0])$
$Lorenzo(2,2,1)$	$H[2,1]-(H[1,1]+H[2,0]-H[1,0])$
$Lorenzo(2,-1,2)$	
$Lorenzo(1,0,2)$	$H[0,2]-H[0,1]$
$Lorenzo(2,1,2)$	$H[1,2]-(H[0,2]+H[1,1]-H[0,1])$
$Lorenzo(2,2,2)$	$H[2,2]-(H[2,1]+H[1,2]-H[1,1])$

6. Footprint

When the data set is large, there may not be enough space to hold it all in main memory. Thus, both the compression and decompression algorithms may need to work from auxiliary storage. In its simplest form, compression estimates the next value using a predictor, then reads the next value from the raw data input stream, encodes the difference, and writes the difference out to the output stream of compressed values. Similarly, decompression estimates the next value using the same predictor, reads in the correction from the input stream of compressed data, decodes it and adds it to the estimated value, and writes the result to the output stream of decompressed values.

Let the term *footprint* denote the amount of main memory needed by the predictor (both during compression and during decompression). The footprint used by the Lorenzo predictor for compressing or decompressing a data set is the size of a single $(n-1)$-dimensional slice, as illustrated in Fig. 3 for the $\mathbb{R}^2$ case and in Fig. 4 for the $\mathbb{R}^3$ case.

The footprint for compressing and decompressing a data set of size D^n is $D^{n-1}+D^{n-2}+\cdots+D+1$. If all the dimensions do not have the same length, the size of the footprint depends on the traversal order, which could be chosen to minimize the size. The footprint is implemented as a circular FIFO queue.

If the corrections are compressed with an adaptive arithmetic encoder, memory to store a probability table is also needed. While this space is generally much smaller than the whole data set, it is often not necessary to store the probability for every possible correction. If memory is scarce, only the frequently occurring, small corrections around zero (Fig. 6) need to be compressed, while occasional large corrections can be flagged and transmitted verbatim, with minimal impact on the compression rate.

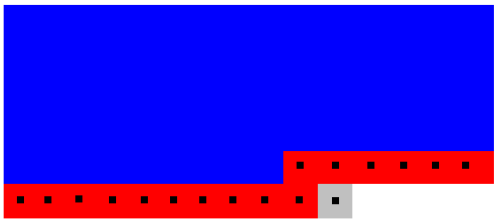

Figure 3: *When compressing an $\mathbb{R}^2$ data set, the next value (grey square) is predicted by using values from the footprint (red). The other previously processed values (blue) are not used by the predictor and need not be kept in memory.*

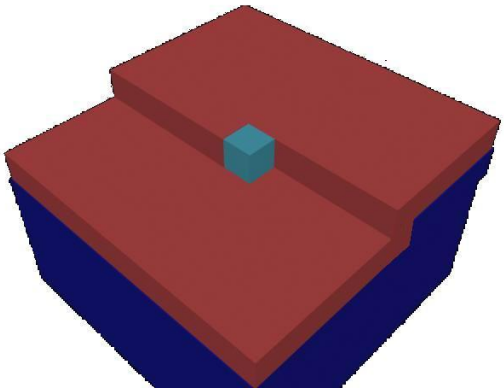

Figure 4: *When compressing an $\mathbb{R}^3$ data set, the next value (light cube in the center) is predicted by using values from the footprint slice (red). The other previously processed values (bottom in blue) are not used by the predictor and need not be kept in memory.*

7. Residual encoding and lossy compression

When lossy compression is acceptable, we allow a small discrepancy between the compressed data and the real data in order to improve the compression ratio. We have considered two different error metrics: L_∞ (maximum error) and L_2 (root mean square error).

To guarantee that we do not exceed a prescribed L_∞ error, we quantize the residuals from our predictor and readjust the values at the scalar field to compensate for any possible error accumulation. Thus the error made is at most the quantiza-

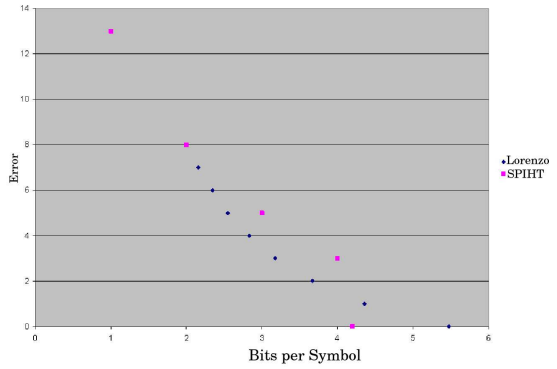

Figure 5: *L_∞ comparison of the Lorenzo Predictor (blue) and SPIHT (pink), a 2D wavelet image compressor.*

tion error. The adjusted corrections are encoded in a lossless fashion.

8. Experimental Results

In this section, we report the benefits of using the Lorenzo predictor as a *preprocessor* to an *arithmetic encoder*. We also compare such a combined approach with wavelet compression. Finally, we demonstrate the impact of the benefits of data coherence on a higher-dimensional predictor.

We have tested our approach both on synthetic and real data. Using synthetic data, we populated a volume data set with functions of higher degree than that of the predictor. For a 3D predictor, when applied to a volume data set filled with a cubic function, the predictor makes a relative error between 3 and 8% (measure taken from applying the predictor to different cubic functions), while a 4D predictor against a volume set filled with a quartic makes a relative error of less than 1%. Both errors are measured in the L_∞ sense.

We have also tested our predictor on two real 4D data sets. After applying the predictor we use two different lossless encoding methods to write the corrections to disk. The first one is to feed the corrections to an adaptive arithmetic encoder. The second one uses a context arithmetic encoder, like the one studied by Bell [7], using the actual prediction as the context for the correction. The context arithmetic encoder gives us a 25% gain over the adaptive arithmetic encoder. We study the benefits of using 4D compression rather than a series of 3D compressed slices. All values are quantized to one byte.

Our first data set, courtesy of Professor Chris Shaw from Georgia Tech, is a 3D model of a house on fire, which shows how the fire, smoke and pressure progress through time and space. This data set has a high number of zones where the scalar values are uniform, and zones that exhibit high gradients, which correspond to the moving front of the fire. Because the high number of 0 corrections in the data set, best results are obtained when using a lossless RLE compression

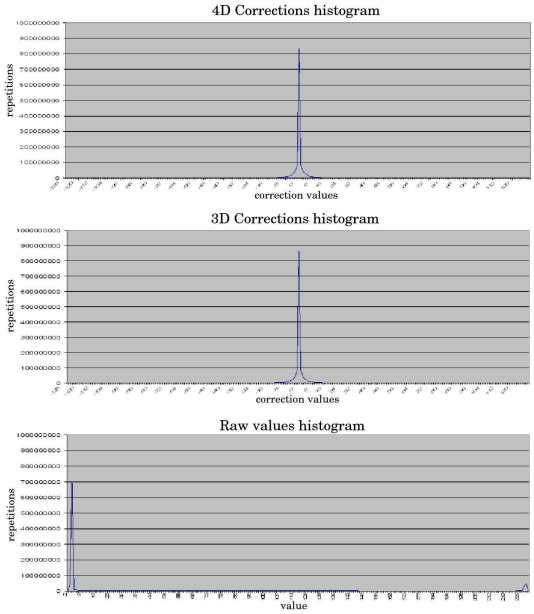

Figure 6: *Histograms for the LLNL data set. Top: Frequency of the 4D corrections (which range from 0 to 1,000,000,000) as a function of their value, ranging from -128 to 127. Middle: Frequency of the 3D corrections for a single time slice. Bottom: Raw values, ranging from 0 to 255.*

method, which we applied both to the 3D and 4D residuals for comparison. The uncompressed 4D data set has a size of 43,202,395 bytes and a total information content (based on its entropy) of 23,491,302 bytes. Compressing each 3D slice independently we obtain 1,076,587 bytes (2.49% of the total), and 524,768 bytes (1.21% of the total) using 4D compression. Due to the high coherence in all dimensions, the 4D predictor outperforms the 3D compression applied to individual slices by 50%.

Our second test data set was produced in a fluid mixing simulation at Lawrence Livermore National Laboratory, as described in [8]. A volume rendered image of this data set is shown in Fig. 1. During the beginning time steps the fluids are virtually at rest and the data set is relatively easy to compress. As the fluids mix up in later time steps, the compression ratio decreases. For this data set, we chose to use an adaptive context arithmetic encoder as a postprocessing step to the Lorenzo prediction.

The 3D time slice predictor produces the best compression on this data set. 3D predictors for 3D slices in all the other directions are less effective. To explain this behavior, consider that this data set was originally computed at high resolution in time (27,000 time steps were simulated), but then decimated and quantized due to limited storage. The floating point data was quantized to one byte and only one frame out of every hundred was actually stored, although no decimation was applied in the spatial dimensions. This

subsampling phase has significantly reduced the coherence along the time axis. Our approach gives us 304,937,058 bytes using the 3D Lorenzo predictor on each time slice (1.77 bits per symbol) and 318,871,620 bytes using 4D prediction (1.85 bits per symbol).

Dataset	4D Lorenzo Predictor	Cubic Wavelets
Smooth 64^4	0.16 Bits/Symbol	0.20 Bits/Symbol
Rough 64^4	3.73 Bits/Symbol	3.28 Bits/Symbol
Rough 128^4	1.75 Bits/Symbol	1.80 Bits/Symbol

Table 1: *Entropy of the residuals produced by wavelets and the Lorenzo predictor for a 4D data set. No quantization or truncation of the data or residuals was done.*

We have compared the Lorenzo predictor with wavelets in two scenarios. In the first case we evaluate lossless compression, where we compare cubic wavelets with the Lorenzo predictor for several 4D data sets. As can be seen in Table 1, which reports the entropy of the corrections of both schemes, wavelets and the Lorenzo predictor produce comparable results. In our second scenario, the Lorenzo predictor was compared to SPIHT [9], an efficient wavelet coder that uses the S+P transform, in terms of rate distortion using lossy compression. Fig. 5 shows that the Lorenzo predictor performs better in the L_∞ sense on this data set. The compression ratios of this example were computed by compressing the residuals using a context arithmetic encoder for the Lorenzo predictor, whereas SPIHT used its hierarchical tree method.

9. Conclusion

The Lorenzo predictor, introduced here, predicts the value of an n-dimensional scalar field F at a sample point v from its $2^n - 1$ previously processed neighbors that form the vertices of an n-dimensional hypercube. The predicted value for $F(v)$ is simply the weighted sum of all values of F at the other corners of the cube. The weights are either $+1$ or -1, depending on the minimal number of cube edges between the sample and v.

The Lorenzo predictor is exact for all polynomials of degree less than n, and its accuracy increases with the smoothness of the data. Because of the limited size of its footprint, the predictor is well suited for out-of-core streaming compression and decompression.

References

1. S. Mennon and M. Rizk, "Large-eddy simulations of three dimensional impinging jets," *Intl. J. Comp. Fluid Dynam.* **7**(3), pp. 275–290, 1996. 1
2. K. Anagnostou, T. Atherton and A. Waterfall, "4D volume rendering with the Shear Warp factorisation," *Symp. Volume Visualization and Graphics '00*, pp. 129–137, Oct. 2000. 2
3. E. Lum, K.-L. Ma and J. Clyne, "Texture hardware assisted rendering of time-varying volume data," *Visualization '01*, pp. 262–270, 2001. 2
4. D. Salomon, "Data Compression: The complete reference," Springer 1997. 1
5. C. Touma and C. Gotsman, "Triangle mesh compression," *Graphics Interface '98*, pp. 26–34, 1998. 1
6. V. Pascucci and R. J. Frank, "Global Static Indexing for Real-Time Exploration of Very Large Regular Grids," *Supercomputing 2001*, Nov. 2001. 1
7. T. Bell, I. H. Witten and J. G. Cleary, "Modelling for Text Compression," *ACM Computing Surveys*, **21**(4), pp. 557–591, Dec. 1989. 5
8. A. Mirin, R. Cohen, B. Curtis, W. Dannevik, A. Dimitis, M. Duchaineau, D. Eliason, D. Schikore, S. Anderson, D. Porter, P. Woodward, L. Shieh and S. White, "Very High Resolution Simulation of Compressible Turbulence on the IBM-SP System," *Supercomputing '99*, 1999. 5
9. A. Said and W. A. Pearlman, "A New Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees," *IEEE Transactions on Circuits and Systems for Video Technology*, **6**(3), pp. 243–250, June 1996. 6
10. S. Guthe and W. Straßer, "Real-time decompression and visualization of animated volume data," *Visualization '01*, pp. 349–356, 2001. 2
11. C. Bajaj , I. Ihm and S. Park, "3D RGB Image Compression for Interactive Applications," *ACM Transactions on Graphics*, **20**(1), pp. 10–38, 2001. 2
12. J. Fowler and R. Yagel, "Lossless Compression of Volume Data," *1994 Symposium on Volume Visualization*, pp. 43–50, Oct. 1994. 2
13. K. Ma, D. Smith, M. Shih and H. Shen, "Efficient Encoding and Rendering of Time-Varying Volume Data," ICASE Report No. 98-22 (NAS/CR-1998-208424), June 1998. 2
14. C. T. Silva, Y.-J. Chiang, J. El-Sana and P. Lindstrom, "Out-Of-Core Algorithms for Scientific Visualization and Computer Graphics," *Visualization '02* Course Notes, Oct. 2002. 2
15. M. Isenburg and S. Gumhold, "Out-of-Core Compression for Gigantic Polygon Meshes," *ACM SIGGRAPH 2003*, to appear. 2
16. T. Chiueh, C. Yang, T. He, H. Pfister and A. Kaufman, "Integrated volume compression and visualization," *Visualization '97*, pp. 329–336, Oct. 1997. 2
17. J. Ho, K. Lee and D. Kriegman, "Compressing large polygonal models," *Visualization '01*, pp. 133–140, 2001. 2
18. MPEG-4. International organization for standardization coding of moving pictures and audio iso//iec jtc//sc29//wg11 n2995, MPEG-4 standard specifications, http://drogo.cselt.it/mpeg/standards/mpeg-4/mpeg-4.html. 2