Preprint
UCRL-JC-151934

Out-of-Core Compression and Decompression of Large n-Dimensional Scalar Fields

L. Ibarria, P. Lindstrom, J. Rossignac, A. Szymczak

This article was submitted to Eurographics 2003, Granada, Spain, September 01, 2003 – September 06, 2003

U.S. Department of Energy

Lawrence Livermore National Laboratory

February 3, 2003

This work was performed 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.

EUROGRAPHICS 2003 / P. Brunet and D. Fellner
(Guest Editors)

Volume 22 (2003), Number 3

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

Lawrence Ibarria[†] Peter Lindstrom[‡] Jarek Rossignac[†] Andrzej Szymczak[†]

[†] GVU Center, College of Computing
Georgia Institute of Technology
Atlanta, Georgia, USA

[‡] Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, California, 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 8192^3 samples [8]. In 4D a typical combustion simulation generated using a High-Performance Parallel Processing Cluster may include 1000 time slices, each representing a regular sampling of a cube at a resolution of 515x512x512 [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 [6].

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 [7] 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 com-

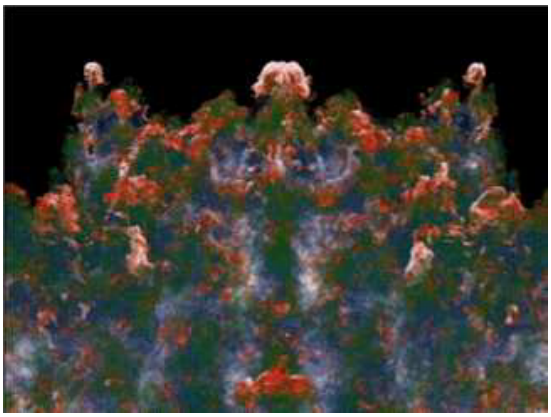

Figure 1: *This is a 4D dataset, from the simulation of two fluids interacting. This image is courtesy of Lawrence Livermore National Laboratory.*

pression in an L_∞ sense, when the residues 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 implementations where the total size of the data set significantly exceeds what can be stored in core memory.

2. Prior Art

A variety of methods to compress 4D volumes have been proposed in recent years. These include wavelets [12], DCT-encoding by Lum [3], RLE encoding [2], and images [4,5]. The wavelet approach uses an interpolating predictor, which, according to informal experiments, produces 50% smaller residues 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 dataset in smaller chunks and compress each one independently. The proposed approach does not require such splitting.

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 recovered. Assume that the data is organized as a regular grid of samples. Both the compression and decompression visit it in scanline order. For simplicity of notation, we use the local coordinate system where sample v has coordinates $(1,1,\ldots,1)$ and its previously visited neighbors are those samples with coordinates in $\{0,1\}^n$. The value of the scalar field $F(v)$ is estimated from the values of the scalar field at the other previously recovered vertices of the n-dimensional unit cube $\{0,1\}^n$ using the following formula:

$$P(v) = \sum_u (-1)^{g(u)} * F(u) \qquad (1)$$

where $P(v)$ is the prediction at v, and $g(u)$ denotes the number of coordinates of u that equal 0. Note that $g(u)$ may also be expressed as $n - u \cdot v$, where n is the dimension 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.

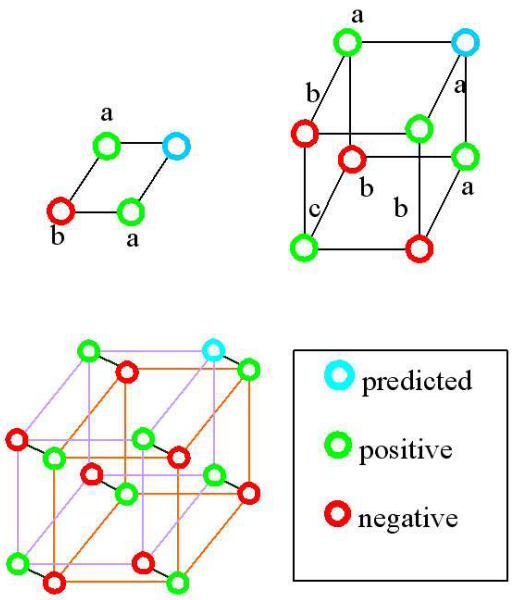

Figure 2: *In the 2D case (top left), the new value is predicted from its neighbors using the parallelogram rule (add the scalar field values at the two a-vertices and subtract the value at the b vertex). In the 3D case, we add the values of the 'a' corners, subtract the values of 'b' corners and add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

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 Q is a polynomial of degree m in n variables with $m < n$ and consider the following theorem and its corollary:

Theorem: For a given monomial M in n variables of degree m inferior to n, the sum of signed values $(-1)^{p(u)}M(u)$ for all the vertices u of the unit cube is zero, with $p(u)$ being the number of coordinates of u that equal 1. Note that $p(u) = n - g(u)$.

More formally, let $u = (x_1, \ldots, x_n)$. A Monomial $M(u)$ has the form: $x_1^{p_1} \ldots x_{k-1}^{p_{k-1}} x_k^{p_k} x_{k+1}^{p_{k+1}} \ldots x_n^{p_n}$, with $\forall_i p_i \geq 0$ and $\sum_i p_i = m$. Then the **theorem** states that $\sum_{u \in \{0,1\}^n} (-1)^{p(u)} M(u) = 0$.

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 $M(x_1, \ldots, x_{k-1}, 0, x_{k+1}, \ldots, x_n) = M(x_1, \ldots, x_{k-1}, 1, x_{k+1}, \ldots, x_n)$. Note that $(-1)^{x_1+\ldots+0+\ldots+x_n} = -(-1)^{x_1+\ldots+1+\ldots+x_n}$. Therefore, the vertices of the cube can be paired so that the values of $(-1)^{p(u)}M(u)$ on the two vertices of a given pair are either both zero or have absolute value one and opposite signs. Thus, the sum of the signed values is zero.

This result may be easily extended to polynomials.

Corollary: For a given polynomial Q in n variables of degree m inferior to n, the sum of the signed values $(-1)^{p(u)}Q(u)$ for all the vertices u of the unit cube is zero, with $p(u)$ being the number of coordinates of u that equal 1.

Proof: Q is the sum of monomials of degree $m < n$. We can permute the two summations. Thus:

$$Q = \sum_i M_i$$

$$\sum_{u \in \{0,1\}^n} (-1)^{p(u)} Q(u) = \sum_{u \in \{0,1\}^n} (-1)^{p(u)} \sum_i M_i(u)$$

$$= \sum_i \sum_{u \in \{0,1\}^n} (-1)^{p(u)} M_i(u) = \sum_i 0 = 0$$

The corollary implies that $(-1)^n M(1, \ldots, 1) = \sum_{u \in \{0,1\}^n - \{1,\ldots,1\}} (-1)^{p(u)} Q(u)$, and hence $M(v) = (-1)^n \sum_u (-1)^{p(u)} Q(u)$ or equivalently $M(v) = \sum_u (-1)^{g(u)} Q(u)$.

The Lorenzo predictor is of highest possible order among all predictors that attempt to predict 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. As a justification, consider the monomial $x_1 x_2 \ldots x_n$ (product of all coordinates). It is zero on all vertices of the unit cube except for one. So, any 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, in two dimensions, the Lorenzo predictor is a linear predictor and hence can reconstruct exactly portions of the scalar field that behave as a linear function $F(x,y) = ax + by + c$.

In $\mathbb{R}^3$ however, 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 cubic polynomials:

$$F(x,y,z,t) = a_1x^3 + a_2x^2y + a_3x^2z + a_4x^2t + a_5xy^2 + a_6xz^2 + a_7xt^2 + a_8xyz + a_9xyt + a_{10}xzt + a_{11}y^3 + a_{12}y^2z + a_{13}y^2t + a_{14}yz^2 + a_{15}yt^2 + a_{16}yzt + a_{17}z^3 + a_{18}z^2t + a_{19}zt^2 + a_{20}t^3 + a_{21}$$

Note that, in the $\mathbb{R}^4$ case, the predictor does not explicitly compute the 21 coefficients of a cubic polynomial, but can nevertheless predict the value of the polynomial from the values it takes at the 15 neighbors of each new sample.

Note that simpler predictors exist that correctly predict all polynomials of degree $m < n$. For example, one may use the n samples that preceed v on a scanline.

Unfortunately, such lower-dimensional predictors are anisotropic and hence 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 $F[xmax, ymax, zmax, tmax]$. The pseudocode for the compression algorithm is presented below:

```
Lorenzo(d, x_1, ..., x_n)
    if all x_1, ..., x_n differ from -1
        p := LorenzoPredictor(d, x_1, ..., x_n);
        Encode(F[x_1, ..., x_n] - p);
    else
        Lorenzo(d - 1, x_1, ..., x_{d-1}, 0, x_{d+1}, ..., x_n);
        for i = 1 to dmax do
            Lorenzo(d, x_1, ..., x_{d-1}, i, x_{d+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, and $dmax$ is the number of samples in the d^{th} dimension. 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 dataset 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 would on the integer lattice be:

Lorenzo$(2,-1,-1)$;
  Lorenzo$(1,-1,0)$;
    Lorenzo$(0,0,0)$; Encodes $H[0,0]$
    Lorenzo$(1,1,0)$; Encodes $H[1,0]-H[0,0]$
    Lorenzo$(1,2,0)$; Encodes $H[2,0]-H[1,0]$
  Lorenzo$(2,-1,1)$;
    Lorenzo$(1,0,1)$; Encodes $H[0,1]-H[0,0]$
    Lorenzo$(2,1,1)$;
      Encodes $H[1,1]-(H[1,0]+H[0,1]-H[0,0])$
    Lorenzo$(2,2,1)$;
      Encodes $H[2,1]-(H[1,1]+H[2,0]-H[1,0])$
  Lorenzo$(2,-1,2)$;
    Lorenzo$(1,0,2)$; Encodes $H[0,2]-H[0,1]$
    Lorenzo$(2,1,2)$;
      Encodes $H[1,2]-(H[0,2]+H[1,1]-H[0,1])$
    Lorenzo$(2,2,2)$;
      Encodes $H[2,2]-(H[2,1]+H[1,2]-H[1,1])$

6. Footprint

When the data 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 for the Lorenzo predictor is the size of a single slice as illustrated in Fig. 3 for the $\mathbb{R}^2$ case and in Fig. 4 for the $\mathbb{R}^3$ case.

Here the red area is the footprint that needs to be in memory, but it is still one dimension less than the whole dataset. For example, a 512^n dataset has a footprint that is 512 times smaller than the whole dataset.

To make use of the small footprint an out-of-core algorithm has to allocate a buffer for storing the footprint points. This buffer has the size of one slice of the data plus the size of the footprint for a data set one dimension less. For example, the size of the footprint buffer in $\mathbb{R}^4$ is $X_M * Y_M * Z_M + X_M * Y_M + X_M + 1$, where X_M is the range in the X axis and so on. The buffer is implemented as a circular FIFO vector of that size.

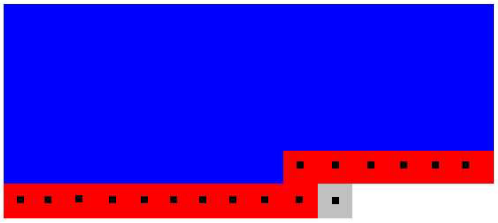

Figure 3: *When compressing a $\mathbb{R}^2$ data set, the next value (grey square) is predicted by using values from the foot print (red). The other previously processed values (blue) are not used by the predictor and need not be kept in memory.*

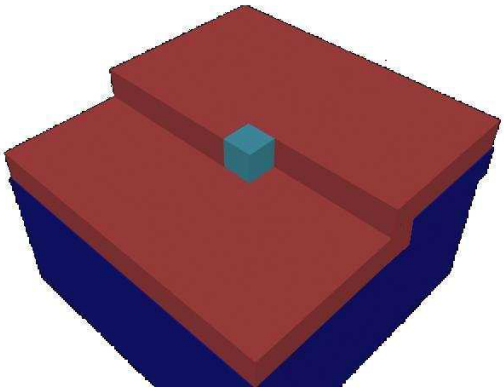

Figure 4: *When compressing a $\mathbb{R}^3$ data set, the next value (light cube in the center) is predicted by using values from the footprint slice (red). The other previously processed values (bottom in blue) are not used by the predictor and need not be kept in memory.*

If the corrections are compressed with an adaptive arithmetic encoder, space to store the probability tables is also needed. This space is much smaller than the whole dataset, and because of the predictor's good behavior, as can be seen in Fig 6, only a subset of all possible corrections need to be compressed with arithmetic encoder, because they represent more than 99% of all corrections though their range is small and centered around 0.

7. Residual encoding and lossy compression

When a lossy compression is acceptable we allow a small discrepancy between the compressed data and the real data in order to improve the compression ratio. For example a 5% tolerance will in practice reduce the size of a compressed file by half. We have considered two different error metrics, L_∞ and L_2. The first is the maximum of the errors in the decompressed version, the latter is the square root of the sum of the square of the errors. So, L_∞ is a metric to be used

when it's imperative that the error never exceeds a threshold. The L_2 error allows a small number of errors to be large while the average error is smaller.

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 much the quantization error. The adjusted corrections are encoded in a lossless fashion.

We have compared two approaches to encoding the residuals. The first one is to feed the corrections to an adaptive arithmetic encoder. The second one is to use a context arithmetic encoder, like the one studied by Bell [9], using the actual prediction as the context for the correction. The context arithmetic encoder gives us a 25% gain over the adaptive arithmetic encoder.

8. Results

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

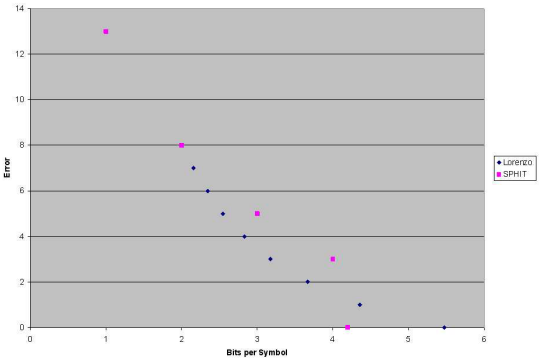

Figure 5: *This is a comparison in L_∞ of the Lorenzo Predictor (blue) and SPIHT (pink), a wavelet implementation, on a 2D dataset.*

We have also tested our predictor on two real 4D datasets. After applying the predictor we use two different lossless encoding methods (discussed below) to write the corrections to disk. We show the improvements in using a 4D compression on top of a series of 3D compressed slices. To keep the size of the datasets small, all values are quantized to one byte.

Our first dataset is courtesy of Prof. Chris Shaw from

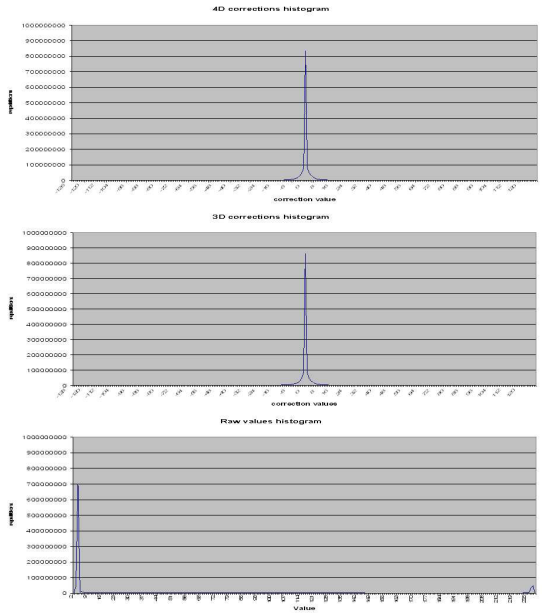

Figure 6: *These three histograms are from our LLNL dataset. The top one shows the frequency of the 4D corrections (which range from 0 to 1000000000), as a function of their value, ranging from -128 to 127. The middle one shows the frequency of the 3D corrections for individual time slices. The bottom one shows the raw values, which range from 0 to 255.*

Georgia Tech. It is the 3D model of a house on fire, showing how the fire, smoke and pressure progress through time. This dataset has a high number of zones where the scalar values are uniform, and zones exhibiting high gradients, which correspond to the moving front of the fire. Because the high number of 0 corrections in the dataset, we use a lossless RLE compression method which we applied both to the 3D and 4D residuals for comparision. The size of the uncompressed 4D dataset is 43,202,395 bytes, with 3D slice compression we reduce it to 1,076,587 bytes (2.49% of the total), and 524,768 bytes (1.21% of the total) with 4D compression. Due to the high coherence in all dimensions, the 4D predictor outperforms by 50% the 3D compression applied to individual slices.

Our second test dataset is a fluid mixing simulation, courtesy of the Lawrence Livermore National Laboratory, identical to the one used in Mirin [10]. This dataset is a simulation of how two fluids interact. In the first time steps the fluids are calm, and the dataset is easy to compress. Later, the fluids mix up and the compression ratio in those frames is much lower. In this dataset we have chosen to use an adaptive context arithmetic encoding as a postprocessing step to the Lorenzo prediction.

The 3D time slice predictor produces the best compres-

sion. The 3D predictors for 3D slices in all the other directions are less effective. To explain this behavior, consider that this original dataset was computed at very high resolution in time. To reduce the storage of this dataset the floating point data was quantized to one byte and one frame out of every hundred was used. This subsampling phase has significantly reduced the coherence along the time axis. Our approach gave us 304,937,058 bytes for 3D (1.77 bits per symbol) and 318,871,620 bytes for 4D (1.85 bits per symbol).

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

Table 1: *This shows the entropy of the residuals computed with wavelets and the Lorenzo predictor for a 4D dataset, in a lossless fashion.*

We have compared the Lorenzo predictor with wavelets in two scenarios. The first is a lossless compression, where we compare cubic wavelets against the Lorenzo predictor for several 4D datasets. As can be seen in Table 1, both Wavelets and Lorenzo predictor have close values, but the Wavelets are slightly better. In the test the entropy of the corrections of both schemes is what Table 1 reports. On our second scenario, the Lorenzo predictor was compared to SPIHT [11], an efficient implementation of wavelets that uses the S+P transform. This compression was lossy, in the L_∞ metric. Fig 5 shows that the Lorenzo predictor performs better in those conditions. The compression ratios of this example were computed by actually compressing the residuals, with a context arithmetic encoder for the Lorenzo predictor and SPIHT using 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 a $\{0,1\}^n$ hypercube of which v is the vertex $(1,\ldots,1,1)$. 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 their minimal number of edges of the cube between the sample and v.

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

We provide the implementation details and illustrate its power on examples of real and synthetic data.

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, October, 2000 2
3. E. Lum, K.-L. Ma and J. Clyne, "Texture hardware assisted rendering of time-varying volume data" *Proc. Visualization'01* pp. 262-270, 2001. 2
4. H.-W. Shen, L. Chiang and K.-L. Ma, "A fast volume rendering algorithm for time-varying fields using a time-space partitioning tree" *Proc. Visualization'99* pp. 371-377, 1999. 2
5. H.-W. Shen and C. Johnson, "Differential volume rendering: A fast volume visualization technique for flow animation" *Proc. Visualization'94* pp. 180-187, 1994. 2
6. D. Salomon, "Data Compression: The complete reference", Springer 1997. 1
7. C. Touma and C. Gotsman, "Triangle mesh compression" *Graphics Interface '98 Conference Proceedings* pages 26-34, 1998. 1
8. V. Pascucci and R.J. Frank, "Global Static Indexing for Real-Time Exploration of Very Large Regular Grids" *Proc Supercomputing 2001*, Nov 2001. 1
9. T. Bell, I. H. Witten and J. G. Cleary, *ACM Computing Surveys (CSUR)*, Volume **21** Issue **4**, December 1989 5
10. A. A. Mirin, R. H. Cohen, B. C. Curtis, W. P. Dannevik, A. M. Dimitis, M. A. Duchaineau, D. E. Eliason, D. R. Schikore, S. E. Anderson, D. H. Porter, P. R. Woodward, L. J. Shieh and S. W. White, "Very High Resolution Simulation of Compressible Turbulence on the IBM-SP System" *Proceedings of Supercomputing 99*, 1999. 5
11. 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*, vol **6**, pp 243-250, June 1996 6
12. S. Guthe and W. StraSSer, "Real-time decompression and visualization of animated volume data" *Proc. Visualization'01*, pp. 349-356, 2001. 2