Spectral Interpolation on 3×3 Stencils for Prediction and Compression

Lorenzo Ibarria
Georgia Institute of Technology
redark@cc.gatech.edu

Peter Lindstrom
Lawrence Livermore National Laboratory
pl@llnl.gov

Jarek Rossignac
Georgia Institute of Technology
jarek@cc.gatech.edu

***Abstract*— Many scientific, imaging, and geospatial applications produce large high-precision scalar fields sampled on a regular grid. Lossless compression of such data is commonly done using predictive coding, in which weighted combinations of previously coded samples known to both encoder and decoder are used to predict subsequent nearby samples. In hierarchical, incremental, or selective transmission, the spatial pattern of the known neighbors is often irregular and varies from one sample to the next, which precludes prediction based on a single stencil and fixed set of weights. To handle such situations and make the best use of available neighboring samples, we propose a local *spectral* predictor that offers optimal prediction by tailoring the weights to each configuration of known nearby samples. These weights may be precomputed and stored in a small lookup table. We show that predictive coding using our spectral predictor improves compression for various sources of high-precision data; through several applications.**

I. Introduction

The acquisition or computation of scientific data sets [1], high dynamic range images [2], and geospatial data [3] usually requires a significant amount of effort and computing resources. Yet, their exploitation is often hindered by the mismatch between the size of the files in which they are stored and the available bandwidth for downloading or visualizing them. Although the loss of precision resulting from controlled quantization or lossy compression may be acceptable for visualization purposes, lossless compression of integer or floating-point values is required in many settings to guarantee the integrity of the data, e.g. when saving state in "restart dumps" for checkpointing numerical simulations [1].

Whereas traditional image compression techniques are capable of lossless compression [4], [5], they were developed for the media industry which usually deals with low-precision data and tolerates trading some accuracy for increased compression. In contrast, we focus on the lossless compression of high-precision data sets represented for example as 32-bit integers or floating-point numbers. The standard approach to lossless compression of such data is based on *predictive coding* [6]–[9], and several prediction schemes for structured data sets have been proposed [4], [10]–[13]. These prior schemes work best when the traversal over the data is simple, e.g. scanline order, so that each sample can be predicted from a single spatial configuration (stencil) of nearby, previously coded samples. When more general traversals are desired or when a nontrivial subset of samples is requested, the configuration of nearby known samples is often irregular and changing, which normally requires falling back on simpler predictors involving fewer samples. In this paper, we address how to make predictions from such irregularly populated neighborhoods that better take advantage of the known samples, and show that such predictors lead to improved compression of high-precision data. Using Fourier analysis, we develop optimal *spectral predictors* for small neighborhoods. While the derivation of the weights for these predictors is somewhat involved, the weights may be precomputed and stored in a small lookup table (available at [14]), and are straightforward to use in a compression scheme.

The compression and streaming approach investigated here follow a simple paradigm: compute the prediction $p_{i,j}$ of the scalar value $f_{i,j}$ as a weighted combination of previously processed samples in the neighborhood $N_{i,j}$; compress the corrections, $c_{i,j} = f_{i,j} - p_{i,j}$, e.g. using entropy coding; and stream them. The paradigm leads to simplicity of implementation, small memory footprint, and excellent compression.

Although our framework is general enough to handle larger neighborhoods and unstructured and higher-dimensional data, we limit our attention in this paper to prediction within 3×3 neighborhoods. (While using larger neighborhoods may improve compression, precomputing and storing the $n(n-1)2^{n-1}$ weights for all possible combinations of known samples in an n-sample neighborhood is impractical for large n.) When the predicted sample is at a corner of a full neighborhood (all eight neighbors known), our spectral predictor reduces to the extrapolating *bi-Lorenzian* predictor; an extension of the previously proposed Lorenzo predictor suited for scanline transmission. When the predicted sample is at the center of a full neighborhood, we obtain the *radial* interpolating predictor, which is four times more accurate than the bi-Lorenzian and is useful in hierarchical transmission. We show that the spectral predictor leads to smaller correctors than other predictors that use a 3×3 neighborhood for lossless compression of high-precision floating-point or integer data. We also explain how to select a priori the best of the nine possible 3×3 neighborhoods that contain the sample to be predicted. This paper is an extension of Spectral Predictors [15]. We include here a novel application of range of isocontour compression with results, and a set of appendices expanding and proving key properties of the Spectral predictors. In our appendices we prove that the sum of the Spectral weights always add to unity, we demonstrate the accuracy of several spectral predictors and their sensitivity to noise, we provide Mathematica code to

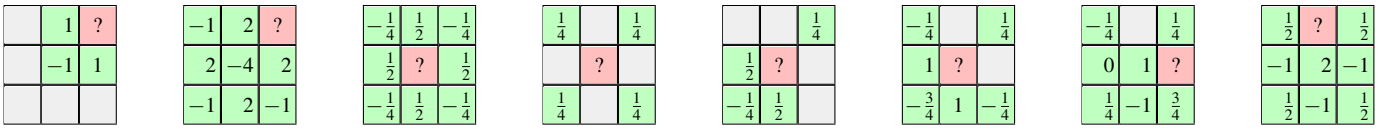


Figure 1. Weights for several spectral predictors used in our experiments: (a) Lorenzo, (b) bi-Lorenzian, (c) radial, (d) bilinear, (e) hybrid linear and radial, (f–h) full spectral.

compute the Spectral weights and we prove that the predictors are self-interpolating.

II. Extrapolating bi-Lorenzian predictor, L^2

Before we derive our spectral predictor, we begin by considering the L^1 Lorenzo predictor [13]. Let f be a one-dimensional function regularly sampled at $\{\ldots, f_{i-1}, f_i, f_{i+1}, \ldots\}$, and let Δ^x be the finite difference $\Delta^x_i = f_i - f_{i-1}$. That is, Δ^x is an approximation of the differential $\frac{\partial f}{\partial x}\,dx$. Setting $\Delta^x_i = 0$, solving for f_i, and substituting L^1_i for f_i, we have as 1D Lorenzo predictor $L^1_i = f_{i-1}$. The Lorenzo predictor extends to 2D via composition of derivatives: $\Delta^{xy}_{i,j} = \Delta^x_{i,j} - \Delta^x_{i,j-1} = f_{i-1,j-1} - f_{i,j-1} - f_{i-1,j} + f_{i,j}$. As the sampling rate of f increases, Δ^{xy} approaches $\frac{\partial^2 f}{\partial x \partial y}\,dx\,dy$ in the limit. Setting $\Delta^{xy}_{i,j} = 0$, we can now express the 2D Lorenzo predictor as

$$L^1_{i,j} = f_{i,j-1} + f_{i-1,j} - f_{i-1,j-1} \tag{1}$$

Thus, in the limit, L^1 correctly predicts all continuous functions f with $\frac{\partial^2 f}{\partial x \partial y} = 0$. In the discrete setting, L^1 recovers linear polynomials, or equivalently bilinear polynomials without highest order term xy. Figure 1(a) shows how the 2D Lorenzo predictor estimates the sample indicated by '?' as a weighted sum of three of its neighbors. We have successfully used L^1 in higher dimensions to predict regular grids [13].

It is natural to ask whether the Lorenzo predictor can be extended to higher-order polynomials that have vanishing higher-order derivatives. To accomplish this, we take finite differences once more and obtain

$$\begin{aligned}\Delta^{xxyy}_{i,j} &= \Delta^{xy}_{i,j} - \Delta^{xy}_{i+1,j} - \Delta^{xy}_{i,j+1} + \Delta^{xy}_{i+1,j+1} \\ &= 2f_{i,j-1} + 2f_{i-1,j} + 2f_{i+1,j} + 2f_{i,j+1} \\ &\quad - 4f_{i,j} - f_{i-1,j-1} - f_{i+1,j-1} - f_{i-1,j+1} - f_{i+1,j+1}\end{aligned}$$

where we define Δ^{xxyy} using central differences. Setting $\Delta^{xxyy}_{i,j} = 0$ and solving for $f_{i+1,j+1}$ we obtain the *bi-Lorenzian* predictor

$$\begin{aligned}L^2_{i+1,j+1} = {} & 2f_{i,j-1} + 2f_{i-1,j} + 2f_{i+1,j} + 2f_{i,j+1} \\ & - 4f_{i,j} - f_{i-1,j-1} - f_{i+1,j-1} - f_{i-1,j+1}\end{aligned} \tag{2}$$

In the limit, L^2 reproduces functions f with $\frac{\partial^4 f}{\partial x^2 \partial y^2} = 0$, and in the discrete setting interpolates biquadratic polynomials without highest order term x^2y^2. Whereas Δ^{xxyy} relates to Δ^{xy} as Δ^{xy} relates to f, L^2 is usually not the successive application of L^1, i.e. in general $L^2_{i,j} \neq L^1_{i,j-1} + L^1_{i-1,j} - L^1_{i-1,j-1}$. Instead, L^2 may be derived by setting to zero the L^1 correction of the L^1 corrections at (i,j). The L^2 weights are shown in Figure 1(b).

The L^1 predictor has been widely used in the image and geometry compression communities [4]–[6], [13]. We are, however, not aware of its extension L^2 having been used for compression of 2D and higher-dimensional data.

III. Interpolating radial predictor, R

In the previous section we presented an extrapolating predictor, L^2, for a corner $f_{i+1,j+1}$ of a 3×3 neighborhood of samples. This predictor arose from the constraint $\Delta^{xxyy}_{i,j} = 0$, a central difference evaluated at the *center* sample of this neighborhood. A more effective predictor is obtained by solving this equation for the function value at the center sample $f_{i,j}$ (the "face sample"), which results in the *radial* interpolating predictor

$$\begin{aligned}R_{i,j} = \frac{1}{4}\big(& 2f_{i,j-1} + 2f_{i-1,j} + 2f_{i+1,j} + 2f_{i,j+1} \\ & - f_{i-1,j-1} - f_{i+1,j-1} - f_{i-1,j+1} - f_{i+1,j+1}\big)\end{aligned} \tag{3}$$

We use the term "radial" to describe this predictor because its weights are radially dependent on the distance to neighboring samples (Figure 1(c)). The prediction $R_{i,j}$ is $2E - C$, where E is the mean of the four edge neighbors $\{f_{i\pm1,j}, f_{i,j\pm1}\}$ and C is the mean of the four corner neighbors $\{f_{i\pm1,j\pm1}\}$. $R_{i,j}$ also equals the mean of the four possible L^1 predictions of $f_{i,j}$.

R has the same predictive power as L^2, i.e. it reproduces biquadratics with no x^2y^2 term, but typically yields better predictions due to the symmetric configuration of its neighborhood. Using Taylor expansion of f one can show that the prediction error of L^2 is $\frac{\partial^4 f}{\partial x^2 \partial y^2}$ (plus higher order terms), which is four times larger than the prediction error for R. Note that to use R, we either must know all eight surrounding neighbors or must estimate them via alternative predictors.

IV. Spectral predictor, S

Our spectral predictor S generalizes L^2 and R to all possible configurations of 0 to 8 known samples and locations of the predicted sample in a 3×3 neighborhood. As in image compression methods based on discrete wavelet [16] and cosine transforms [17], we capitalize on the fact that the signal power is often heavily skewed towards low frequencies. In frequency transforms, this results in small, compressible high-frequency detail coefficients, whereas in predictive coding "smooth" interpolants recover most of the low-frequency response, leading to small correctors for the missing high-frequency content.

In this section, we design as-smooth-as-possible interpolants for irregular sample configurations. We seek to eliminate or, when not possible, to minimize high-frequency responses in

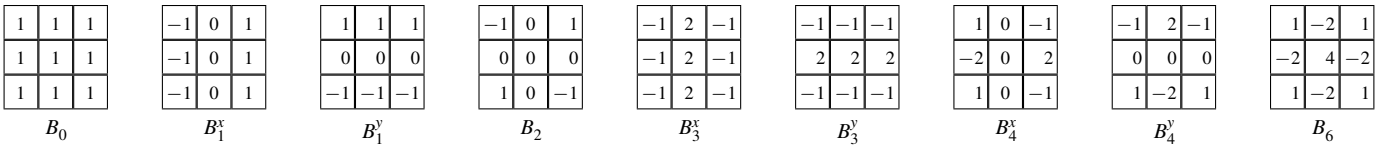


Figure 2. Basis functions for the 2D discrete cosine transform (not normalized).

the interpolant. The resulting predictors and their sets of weights can be stored in a lookup table indexed by the mask of known and unknown values and the location of the predicted sample.

We build upon the work by Isenburg et al. [18], who use the Fourier transform to predict the geometry of n-sided polygons to be "as regular as possible" given $m < n$ known vertices. They express the vertex coordinates of the polygon in the complex plane, apply the discrete Fourier transform (DFT) to this n-vector of consecutive vertex coordinates, set the highest $n-m$ frequencies to zero, and compute the inverse transform to obtain the complex coordinates of the predicted vertices. Because the Fourier transform is linear, the unknown vertices can be expressed as a linear combination of the known vertices. By working out the mathematics of the forward and inverse Fourier transforms, one can a priori establish a set of weights for a given configuration (m,n) of known and unknown number of vertices (i.e. the weights are not dependent on the geometry of the known samples). Because Fourier frequencies come in pairs, this approach works well when m is odd as then the resulting weights are guaranteed to be real. One can show that the discrete cosine transform (DCT) can instead be used when m is even. Lifting the DFT to higher dimensions, Isenburg et al. further showed that the L^1 predictor from Section II is in the spectral sense the optimal predictor (i.e. smoothest interpolant) for hypercube-like neighborhoods with one unknown sample.

We begin by extending the general approach of Isenburg et al. to 3×3 neighborhoods to re-derive the bi-Lorenzian and radial predictors and show that they are optimal. We will make use of the two-dimensional (orthonormal) discrete cosine basis

$$\begin{gathered}\{u(x)u(y),\ s(x)u(y),\ u(x)s(y),\\ s(x)s(y),\ c(x)u(y),\ u(x)c(y),\\ s(x)c(y),\ c(x)s(y),\ c(x)c(y)\}\end{gathered}$$

where x and y vary over the domain $\{-1,0,+1\}$ of our 3×3 neighborhood, and where

$$u(x)=\sqrt{\tfrac{1}{3}},\quad s(x)=\sqrt{\tfrac{2}{3}}\sin\left(\tfrac{1}{3}\pi x\right),\quad c(x)=\sqrt{\tfrac{2}{3}}\cos\left(\tfrac{2}{3}\pi x\right)$$

The cosine basis is shown un-normalized in Figure 2. We unfold the 3×3 matrix into a single 9-dimensional vector $b=[f_{i-1,j-1}\quad f_{i,j-1}\quad f_{i+1,j-1}\quad\cdots\quad f_{i+1,j+1}]^T$ of sample values, and write the cosine basis as a 9×9 orthogonal matrix B, where the columns of B are the basis functions. Then the forward discrete cosine transform is simply $x=B^Tb$, with x being the DCT coefficients in order of increasing frequency.

To extend the ideas of Isenburg et al. from 1D to 2D, we must rank the basis functions by increasing frequency. The cosine basis formulation gives us pairs of frequencies (ν_x,ν_y) for the horizontal and vertical direction, which must be consolidated into single frequencies. We approach this by deriving the cosine basis through eigenanalysis of the symmetric combinatorial graph Laplacian $\mathcal{L}$ (also called the Kirchoff matrix [19])

$$l_{ij}=\begin{cases}\deg(i) & \text{if } i=j\\ -1 & \text{if } i \text{ and } j \text{ are adjacent}\\ 0 & \text{otherwise}\end{cases}\tag{4}$$

where we consider the graph formed by the 3×3 neighborhood in isolation, with vertical and horizontal edges between adjacent samples. Here $\deg(i)$ denotes the degree or number of neighbors of a sample i, e.g. $\deg(i)$ is 2 for corner samples, 3 for edge samples, and 4 for face samples. As noted by Taubin [20], the eigenbasis of the normalized (asymmetric) Laplacian coincides with the Fourier basis, and the eigenvalues $\{\lambda_i\}$ of $\mathcal{L}$ correspond to frequencies. The above un-normalized (symmetric, positive semidefinite) Laplacian $\mathcal{L}$ has real non-negative eigenvalues $\{0,1,1,2,3,3,4,4,6\}$ and the cosine basis as eigenbasis. We will use B_λ to denote the eigenvector (i.e. basis function) with corresponding eigenvalue λ, and B_λ^x and B_λ^y to distinguish pairs of eigenvectors with equal eigenvalues (Figure 2).

Our formulation shows that there is a unique highest frequency $\lambda=6$ with associated basis function B_6. Given the eight known samples in the bi-Lorenzian and radial predictors, similarly to Isenburg et al., we set the highest frequency response x_6 to zero and solve for the unknown sample as a linear combination of the $m=8$ known samples, which results in the weights given in Equations 2 and 3 for corner and center predictions. When $m<8$, a similar strategy is possible by zeroing $9-m$ of the highest frequencies. However, we may need to resolve two issues: (1) The $9-m$ first basis functions may not form a basis for the set of known samples, e.g. $\{B_0,B_1^x,B_1^y\}$ is not a basis for $b=[f_{i-1,j-1}\quad f_{i,j-1}\quad f_{i+1,j-1}\quad 0\quad\cdots\quad 0]^T$. (2) In situations when only one of B_λ^x and B_λ^y is needed (e.g. when exactly two samples are known, as in Figure 3), we may reduce the total frequency response by choosing a linear combination of B_λ^x and B_λ^y.

Let M be an $m\times n$ mask matrix that extracts the m known samples Mb from b, i.e. each row of M has a single one entry and remaining zeros. We wish to solve the underconstrained system $MBx=Mb$ for x with as many high frequencies of x zeroed as possible. This can be done via linearly constrained least-squares methods, which involves symbolic inversion of an $(m+n)\times(m+n)$ matrix. We show here how to accomplish the same goal via inversion of a smaller $m\times m$ matrix.

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$P^T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

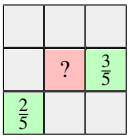


Figure 3. Example mask matrix M, interpolation matrix P, and predictor weights.

We first must find an m-dimensional basis for Mb by selecting from or combining the $n > m$ column vectors MB; any excluded vector from MB will implicitly have its corresponding frequency response zeroed. Our approach is to incrementally construct an $n \times m$ interpolation matrix P that linearly combines vectors from MB such that $MBPy = Mb$ is a fully constrained system of m equations, with $Py = x$. We achieve this by adding to P columns that select basis functions from MB in order of increasing frequency λ. If a basis function projected onto the space of known samples is redundant (linearly dependent) with respect to the partially constructed basis, we exclude it and consider the next basis function. When we encounter an eigenspace, i.e. two basis functions with the same eigenvalue, one of three situations arises: (1) The whole eigenspace is redundant, and we exclude it. (2) The whole eigenspace is nonredundant, and we include it. (3) The eigenspace is partially redundant, in which case we first "rotate" the eigenspace by an angle θ to make one of the rotated and projected basis functions redundant. (Note that any rotation of an eigenspace preserves eigenvalues and orthogonality with the rest of the basis.) This leaves a nonredundant basis function $B_\lambda^\theta = \cos(\theta)B_\lambda^x + \sin(\theta)B_\lambda^y$ and we add to P a column that has $\cos(\theta)$ and $\sin(\theta)$ in the rows corresponding to B_λ^x and B_λ^y. The effect of this rotation is to "align" the basis function with the spatial configuration of known samples. One can show that this rotation leads to the minimal total frequency response $||x||$.

We may now compute $x = P(MBP)^{-1}Mb$ using matrix inversion. We are, however, not interested in the frequency response x but in the weights of the known samples Mb. Hence we apply the inverse DCT to x and compute $Bx = Wb$, where W is the $n \times n$ weight matrix $W = BP(MBP)^{-1}M$, which is then used to predict unknown samples in b from known ones.

We have implemented this method symbolically in Mathematica. A complete code listing is found in Appendix I. Computing exact weights W for all neighborhood configurations results in 41 unique weights in the range $[-4, +4]$ that are predominantly integers and otherwise rationals (see [14] for a complete list of weights). In Appendix IV we prove that our weights always add to one, which makes our scheme affine invariant. Another desirable feature of spectral interpolation is self-interpolation: interpolating a sample s from a set S and then interpolating t from $T = S \cup \{s\}$ yields the same result as interpolating t directly from S. We prove this property in Appendix V.

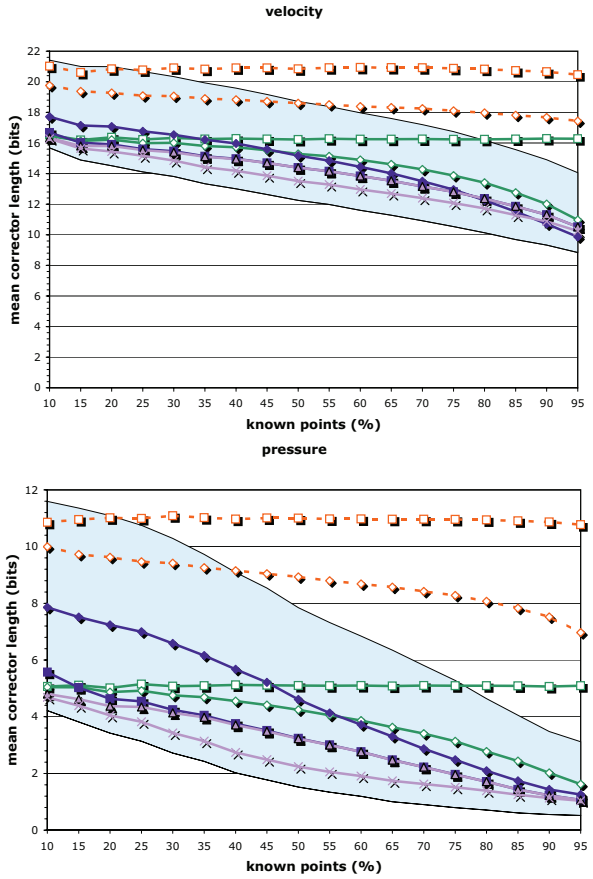


Figure 4. Predictor quality as a function of number of known points in the 3×3 neighborhood. The shaded area indicates the range between best and worst spectral prediction.

A. Choosing a neighborhood

Via translation we can form nine 3×3 neighborhoods around each predicted sample p. Depending on the configuration of known samples it is not immediately clear which neighborhood to predict from. We propose training the compressor on the given data set: each of the 9×2^8 predictors is exercised on each sample and receives a ranking based on the mean error it makes. This short ranking is transmitted before compression begins and determines the choice of predictor. In Figure 4 we show using random sampling of two data sets that our approach improves upon several alternatives that we have explored, including the neighborhood centered at p and the mean or the median of all nine predictions. For calibration, we also report the results for the best (lowest residual) of the nine neighborhoods (which unfortunately is not available to the decoder), as well as the mean and median of constant (single-value) and L^1 prediction.

V. Applications of Spectral Prediction

Our spectral predictor is particularly useful in applications where standard compression techniques, e.g. based on wavelets, are not practical, such as for encoding data sets with irregular domains due to manual or automatic extraction, in-

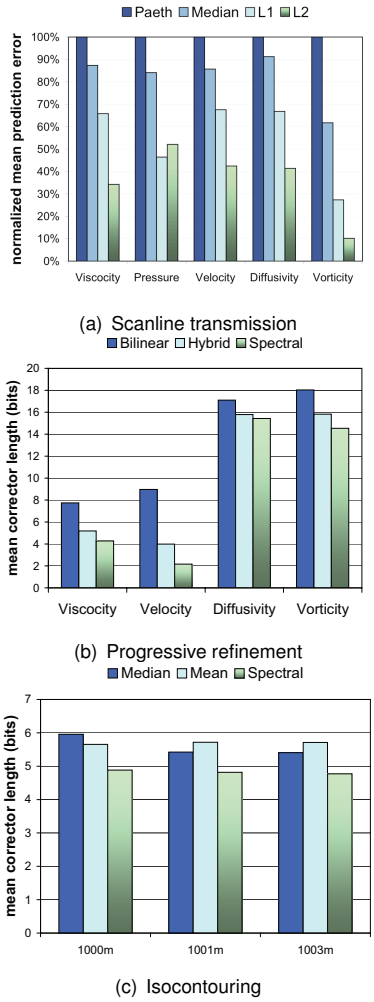


Figure 5. Prediction results for three different applications.

painting, selective updates, adaptive sampling, or range queries that extract those samples whose values fall within an interval. Irregular sample configurations also arise when the data is traversed in other than scanline order, or in mesh compression, where the domain connectivity is inherently irregular. For lack of space, we here consider only a few of these applications.

We evaluate predictor performance in terms of the number of significant corrector bits, which is the dominating cost in predictive coders for high-precision data [7], [8], including our own [9]. For floating-point data, we compute an integer corrector that measures the number of distinct floating-point values between the actual and predicted value (see [9]).

A. Scanline transmission

The most straightforward method to compress regularly gridded data is to make a scanline traversal, e.g. row-by-row from bottom to top and from left to right within each row. We here compare our bi-Lorenzian L^2 predictor with other scanline predictors proposed for image compression: the Paeth predictor [12] used in the PNG image format [5], the median predictor used in JPEG-LS [4], and the L^1 Lorenzo predictor [13] used by Lindstrom and Isenburg [9]. All except

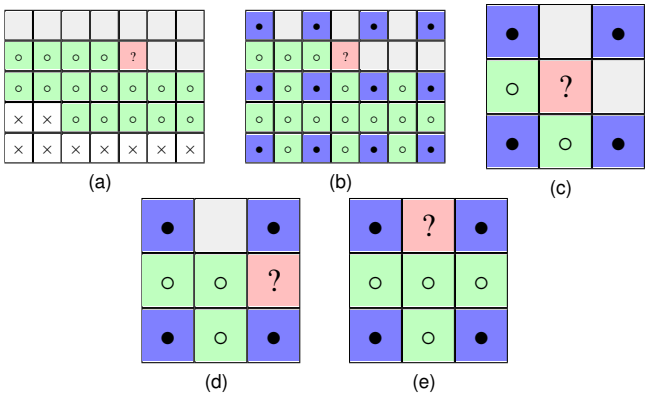


Figure 6. (a) L^2 footprint (circles) maintained during scanline traversal. (b) Coarse-resolution (solid) and fine-resolution (hollow) processed samples in a hierarchical traversal. Within each level of resolution, scanline traversal is used, resulting in three predictor stencils: (c) face, (d) vertical edge, and (e) horizontal edge sample.

L^2 predict a sample from the same set of three neighbors (Figure 1(a)).

In order to apply L^2 in a scanline traversal, two rows of previously coded samples must be maintained (Figure 6(a)). To bootstrap the predictor, one may use lower-dimensional Lorenzo prediction to first recover domain boundaries. Alternatively, one may use the spectral predictor for partially known neighborhoods described in Section IV.

Figure 5(a) shows the results of predicting multiple 2D slices of the single-precision floating-point scalar fields shown in Figure 8 obtained from a fluid dynamics simulation [1]. On high-precision data like this, L^2 often offers substantially better prediction than predictors that use smaller stencils. The benefit of a larger stencil comes at the expense of higher sensitivity to quantization, however, due to accumulation of per-sample errors and larger (in magnitude) weights. Analysis shows that the prediction error due to quantization is three times larger for L^2 than for L^1. Hence L^2 generally performs worse than L^1 on low-precision data such as 8-bit images.

B. Progressive refinement

Often, data sets are transmitted progressively, doubling the resolution in x and y after each refinement. The missing values within a refinement level may be transmitted in scanline order, as shown in Figure 6(b), which results in three 3×3 neighborhood configurations from which samples are predicted (Figure 6(b–d)).

We consider three predictors for the face sample (Figure 6(c)): bilinear interpolation B of corner samples (Figure 1(d)), spectral prediction S_f (Figure 1(f)), and a hybrid predictor H (Figure 1(e)) that first linearly interpolates the unknown neighbors at the vertical and horizontal edges from their immediate neighbors to fill the neighborhood and then predicts the face point using radial prediction R. Note that both B and H are instances of spectral prediction that simply ignore some of the known neighbors. For the edge points, B and H resort to linear interpolation of corner points for prediction (since no other reasonable non-spectral predictor is available),

while our spectral predictor is able to make use of all decoded samples (Figures 1(g) and 1(h)).

Figure 5(b) illustrates the advantage of using all known neighboring samples in the prediction. S_f offers in all cases superior prediction over B and H, leading in one case to as much as a 4:1 improvement in compression. Note that one may choose a different traversal order within each level. In fact, our experiments show that transmitting the missing edge samples first and then the face samples further improves compression, in part because the face samples may be predicted using the radial predictor with fully known (not simply estimated) neighborhoods.

C. Isocontouring

In many scientific, engineering, and medical applications, regularly sampled volumetric scalar fields are visualized in terms of isosurfaces. For instance, a remote viewer may wish to see the isosurface $S(t)$ formed by all points at temperature t or to explore the family $S(T)$ of isosurfaces with temperatures in a range $T = [t_{min}, t_{max}]$. Instead of transmitting the geometry of $S(t)$ or some compressed form of its animation, it is often more effective to transmit the minimal subset of scalar values needed to reconstruct the single isosurface $S(t)$ or family of isosurfaces $S(T)$ [21]. To satisfy this query, one needs to transmit not only the samples with values in T, but also some of their neighbors to obtain a complete "scaffold" around the surface. In a scenario where the the remote user later decides to extend T to a larger interval, compression and incremental transmission of the subset of additional samples would often be preferable over complete retransmission. For both initial and incremental transmission, it is not obvious how to predict the irregular subset of sample values using traditional means.

We transmit the minimal subset of values that completely determine the isoset requested by the client. Our approach is based on the traversal, identification and transmission of nodes and their associated value. Although the client might extract edges or triangles from the nodes to visualize the isoset, this task has no effect on transmission cost.

Our isosurface generation (extraction) method is very similar to that of marching cubes. Doing an exhaustive search of the regular grid is not convenient. We assume that a precomputed a set of seeds so we only access the cells intersected by the isosurface.

The encoder and decoder perform the same traversal hence we discuss them together. Our approach traverses each component of the isocurve. It starts with a seed stick and encodes/decodes its two node values (A, B from Figure 7).

The encoder and the decoder process need to store in memory the values at the nodes that determine the isosurface. Our implementation uses a quadtree to store the nodes and their values. This structure provides fast, adaptive storage, while maintaining spatial relationships between the nodes.

Each cell is processed equally, from a stick and two known values (see Figure 7), our approach traces the isocurve and determines which link is the stick the curve exits through. While the isocurve never bifurcates, it is possible for a cell to have four sticks, the isocurve traversing the cell twice. In such case, we use a heuristic to select the behavior of the isocurve inside the cell.

The two nodes at the entrance stick are known to both the encoder and the decoder, but the other two nodes of the cell might not be known (see Figure 7).

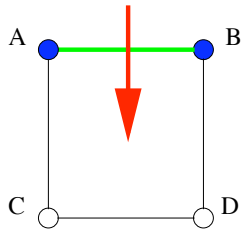
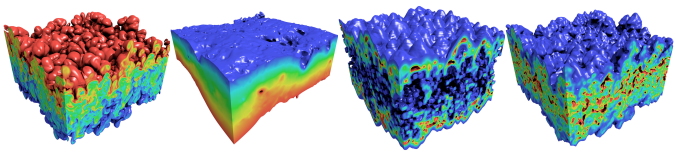


Figure 7. Example of a cell in 2D. Green is the entrance stick, blue represent two already encoded nodes, and red is the traversed part of the isosurface.

There are three possible cases when we process a cell (the rest are rotation and mirroring of these cases):

1) *Three nodes from the cell are known*, the two nodes from the entrance stick and another. Without loss of generality, we assume that C is the other known point. If $\overline{AC}$ is a stick, then $\overline{AC}$ becomes the exit stick. If it is not a stick, we encode/decode D, and test to see which edge is a stick, $\overline{BD}$ or $\overline{CD}$. The edge that is a stick will become the exit stick.
2) *Only the two nodes from the entrance stick are known*. We use our prediction to guess the values at the remaining nodes: C' is the prediction for C and D' is the prediction for D. If $\overline{AC'}$ is a stick, we encode/decode the node C, and this becomes case 1. If it is not a stick, we check $\overline{BD'}$ for stick. If it is, we encode/decode D and go to case 1. If neither $\overline{AC'}$ nor $\overline{BD'}$ is a stick, we compute the vertex from the entrance stick; the point where the isosurface intersects the entrance stick. We encode/decode then the point closest to the vertex. If the vertex was closer to B than to A, we would encode/decode the node D. We proceed with case 1.
3) *All nodes in the cell are known*. If there is one single other stick, that becomes the exit stick (nothing needs to be encoded). There are cases when every edge is a stick. We call that kind of cell an *x-cell*. X-cells are ambiguous, there are two possible correct solutions to the trace of the isosurface in an x-cell (see Figure 9). We proceed in a similar way as case 2. We compute the vertex at the entrance stick. If the vertex is closer to B than to A, $\overline{BD}$ becomes the exit stick.

We disambiguate using a heuristic, we use the node closer to the vertex of the entrance stick to determine what stick to predict or to trace the curve to. Other heuristics that might be used here need trace of the isocurve previous to the entrance of the current cell, and extrapolate the path of the curve. It is important to note that we encode only the values at nodes that define the isocontour, the process of extracting the isocontour, and thus resolving the ambiguities, is left for



(a) Density (b) Pressure (c) Diffusivity (d) Viscocity

Figure 8. Interval-volume renderings of some of the 3D scalar fields used in our experiments.

the client application and it is independent of our proposed scheme.

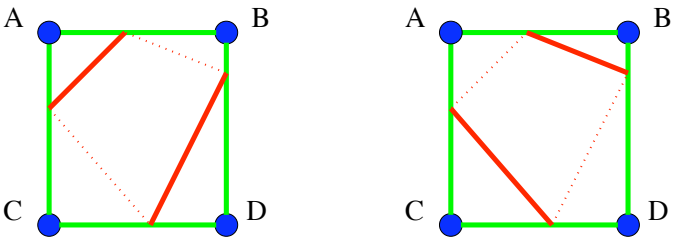


Figure 9. Ambiguous case in 2D, *x-cells*. Both isosurfaces are correct.

To encode the value of a node, we use Spectral prediction. The structure where the nodes and values are stored, the quadtree, provides a mask specifying which values are known. We consider a 5×5 neighborhood around the predicted node. The neighborhood for our Spectral prediction is 3×3; we can apply nine different spectral predictions. The 3×3 neighborhood with more known values is chosen to be the applied spectral prediction.

Using the real value and the prediction, we encode the residual as described by Isenburg and Lindstrom [9].

The position of the seeds is not known to the decoder, the seeds are transmitted in a different stream in raw format. For large isosurfaces, the ratio of seeds to encoded values is small. A seed is encoded as a link of the regular grid that intersects the isosurface. A seed is encoded as the position of a node of the link and symbol describing in which direction the second node of the link lays. In 2D, the discriminating symbol needs only one bit. The start of the decode process with a seed needs to bootstrap the traversal process by decoding the two nodes incident to the seed stick. In the case that there is no nearby point to predict the first node, we use the isovalue; the node's value is bound to be close to the isovalue. The second node already uses Spectral prediction. In the case that only the first node is known with value w, we linearly extrapolate the second node's value using the isovalue v; the prediction is $2v - w$.

It is important to point out the fact that our approach never sends the value of the same node twice. If a previous isocurve encoding had transmitted a node value, all following isocurves that need such node will have it available locally and will not need to encode it again.

We have tested our approach on two datasets. The first dataset is a distance map, its isocontours take the shape of a circle. A few renderings of that dataset are in Figure 12. We use the scientific dataset vorticity from Cook et al [1]. It has

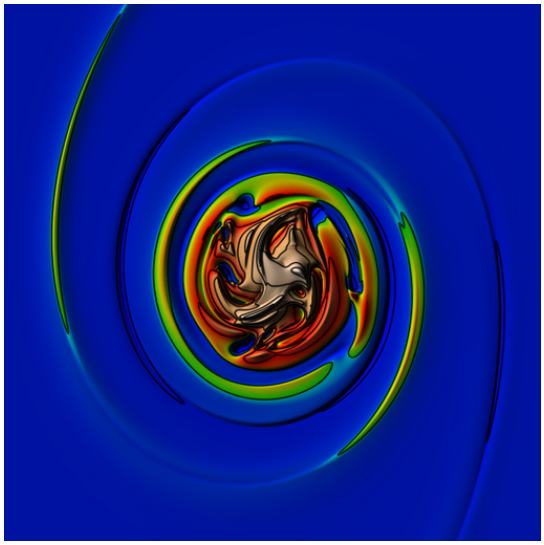


Figure 10. Vorticity dataset. 2D regular grid of 1025×5000, from [1]. This is a snapshot of a fluid simulation.

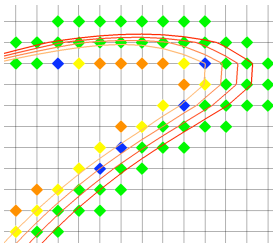
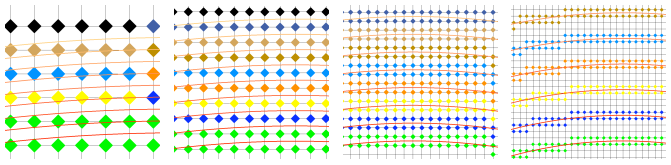


Figure 11. Zoom in a set of decompressed isosurfaces. The nodes transmitted with each isosurface are colored different.

1025×5000 size. It is the product of a fluid simulation. We have rendered it using white for the highest value and black for the lowest, see Figure 10. We use the circle dataset as a proof of concept of our ideas, and we use the vorticity dataset to show the effectiveness of our technique with real world data.

We transmit the nodes defining an isocurve, but it is common to refer to the compression of isocurves by the number of vertices the isocurve has. From here on, we refer to *bits per node* to the average number of bits used to encode the node values transmitted, and *bits per vertex* to refer to the average number of bits per vertex of the transmitted isocontour.

Our technique can be used to compress one single isosurface. The performance achieved in that case should not be compared to regular isosurface compression because we encode more information than a single isosurface. For the circle dataset, our technique achieves 17.13 bits per vertex. Considering that the circle is a smooth dataset, that value is rather high. Our method achieves 12.1 bits per node, that is considered normal for a smooth dataset. In the 2D case, while compressing a single isosurface we use an average of 4.2 known neighbors. Due to our 3×3 neighborhood and the fact that the curve of the circle goes forward, we apply most of the time a prediction that is linear.



(a) small (b) medium (c) large (d) very large

Figure 12. From our synthetic circle dataset, four set of isosurfaces. From left to right, the step between isovalues is doubled. Each isocontour is colored with a different color, the points drawn are transmitted with each isocontour. It can be seen how closer isocontours transmit less points.

Our method's strength shows when we compress several isocontour. Figure 12 shows four set of isocontour. As can be seen in Figure 13, when compressing a set of isocontour that are very far apart from each other, the total result is just the compression of several single isolated isocontours. As the distance between the different isocontours draw closer, as seen in the *large*, *medium* and *small* set of isocontours, the cost of transmitting each new isosurface is reduced. The number of average neighbors used in prediction was 6.5, 7.3 and 7.4 for the datasets from *large* to *small*. The ratio between nodes and vertices was 0.5, 0.7 and 1.2 respectively. The *large* dataset does have almost the same node/vertex ratio as the individual isocontour, but its prediction is improved by having nearby values from previously transmitted isocontours. As long as the new isocontour is nearby already decompressed data, our technique offers improvement. The total compression bits per vertex for the *very large*, *large*, *medium* and *small* is 17.28, 13.64, 6.49 and 4.13 respectively.

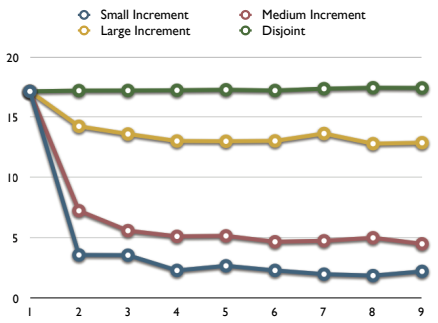


Figure 13. Bits per vertex for each set of isosurfaces from Figure 12. The x axis is the number of the transmitted isosurface. This shows the improvements of transmitting isosurfaces that overlap with previously decoded data.

As a result of the way the nodes are predicted, the order in which isocontours are encoded affects the compression ratio. For sets of disjoint isocontours the order has no influence on compression, but for sets of close isocontours it does affect the total compression ratio. For the case of the *medium* set of isocontours, we have ordered them to illustrate an extreme worse case scenario: the first half of the isocontours are transmitted in an order where there is no overlap, even though there are enough nearby points to help prediction, and the second half are transmitted but few nodes need to be send. The first half have results similar to the *large* set of isocontours, the second half has near perfect prediction (an average of 8.0 used neighbors), 0.8 bits per vertex. The total compression cost for the incremental *medium* set of isocontours is 6.49 bits per vertex, the cost of our order is 8.22. Order matters when compressing a dataset, on the other hand, the order is defined by the user.

For the vorticity dataset, from on a scientific simulation of interacting fluids, we have tested two configurations. We call it *medium* and *large*. The circle dataset was homogeneous, the vorticy dataset has areas with more slope and with less, comparisons between sets of isocontours cannot be made on equal footing. Our *large* isocontour range has its first isovalue at 5.5278, contains 9 isocontours with an isovalue increment of 1.0 for each isocontour. The *medium* dataset starts at the same isovalue, but the increment is 0.5 in this case. We achieve a total compression of 16.6 bits per vertex on the *large* isocontour range and 12.08 for the medium isocontour range. The bits per vertex we obtain for a single isosurface is 33.97, which tells of the harshness of the dataset. Using extrapolating predictors with a reduced number of known points (close to 4 on our 2D tests) is ineffective for the lossless compression of floating point data. This causes us to achieve subobtimal results in single isocontour transmission. The following isocontour transmission will reduce the average bit per vertex.

We have tested encoding the previous set of isocontours in different orders (see Table I). In all cases we have observed that if the objective is to encode all the nodes in a range, it is an improvement encoding them sequentially rather than hierarchically. In the sequential encoding we have appreciated a 20% improvement over hierarchical encoding. The improvement is dependent on the accuracy of prediction. Sequential encoding encodes each point with a close-to-full stencil (7 to 8 neighbors), while hierarchical encoding is relegated to use stencils scarcely populated ((4 neighbors on average) for half the encoded isocontours.

TABLE I.
TOTAL COMPRESSED FILES FOR THE SAME SET OF ISOCONTOURS FOR THE 2D CIRCLE, IN DIFFERENT ENCODING ORDER.

	Total File Size
Medium {1,2,3,4,5,6,7,8,9}	32537
Medium {1,3,5,7,9,2,4,6,8}	41214
Medium {1,5,9,3,7,2,4,6,8}	44599

We also report experiments of extracting isolines in 2D from the Puget Sound 16-bit terrain surface (available at [22]). We first extracted an isocontour at 1000m elevation and predicted all necessary samples, then incrementally transmitted missing values for isocontours at 1001m and 1003m, resulting in an average number of known neighbors of 5.13, 5.42, and 5.41, respectively. Since samples are often not available for predictors like L^1 to be applied, we compare our spectral predictor with predictions based on the mean and median sample value in a 3×3 neighborhood centered on the predicted sample. We observed consistent reduction in corrector bit

length (13–22%) using the spectral predictor, even for this lower-precision data set (Figure 5(c)).

VI. Conclusions

We propose two new predictors, the bi-Lorenzian L^2 and the radial R, which predict the value of a sample f from eight values in a 3×3 neighborhood of which f is the corner (for L^2) or the center (for R). More importantly, we propose the spectral predictor S, which extends L^2 and R to all configurations of 0 to 8 known samples and locations of f in a 3×3 neighborhood. We argue that S is the best predictor from a 3×3 neighborhood, provide a strategy for selecting the most promising neighborhood that contains f, and demonstrate the benefits of S over competing predictors in three simple applications.

While applied only to 3×3 neighborhoods in 2D regular grids here, our framework, which is based on the eigenstructure of the combinatorial graph Laplacian, easily generalizes to higher dimensions and to irregular grids. One immediate application we envision is geometry prediction for polygonal and polyhedral meshes. In the more general setting, multiple and larger neighborhoods may arise, possibly leading to very large weight lookup tables. In order to reduce memory requirements, symmetry, non-uniqueness of weights and weight combinations, and unity constraints can be exploited, however having a more efficient procedure for on-demand computation of weights than symbolic or even numerical matrix inversion is clearly desirable.

Appendix I
Implementation

Listing 1 is a Mathematica 5.0 implementation of the spectral interpolator. Given a Laplacian matrix $\mathcal{L}$ that represents the stencil from which predictions are made and a vector S that specifies which samples are known (1) and unknown (0), the weight matrix W is computed. The bi-Lorenzian weights, e.g., are obtained by weight[{0, 1, 1, 1, 1, 1, 1, 1, 1}][[1]]. For completeness, the 3×3 Laplacian is included here, but it may be redefined for arbitrary stencils.

Listing 1. Mathematica code for computing spectral weight matrix.

```
Needs["LinearAlgebra`Orthogonalization`"]

(* Laplacian for 3x3 stencil *)
lap = {{  2, -1,  0, -1,  0,  0,  0,  0,  0 },
       { -1,  3, -1,  0, -1,  0,  0,  0,  0 },
       {  0, -1,  2,  0,  0, -1,  0,  0,  0 },
       { -1,  0,  0,  3, -1,  0, -1,  0,  0 },
       {  0, -1,  0, -1,  4, -1,  0, -1,  0 },
       {  0,  0, -1,  0, -1,  3,  0,  0, -1 },
       {  0,  0,  0, -1,  0,  0,  2, -1,  0 },
       {  0,  0,  0,  0, -1,  0, -1,  3, -1 },
       {  0,  0,  0,  0,  0, -1,  0, -1,  2 }}

(* eigenbasis B of Laplacian *)
b = Transpose[GramSchmidt[Reverse[Eigenvectors[lap]]]]

(* ordered eigenspace index sets *)
e = Split[
  Range[Length[lap]],
  Equal @@ Reverse[Eigenvalues[lap]][[{##}]] &
]

(* mask matrix M(S) *)
mask[s_] := Select[DiagonalMatrix[s], Norm[#] > 0 &]

(* add linearly independent columns to matrix P *)
extend[mb_, {}, qt_] := qt
extend[mb_, pt_, qt_] := Join[
  pt,
  Select[
    RowReduce[
      NullSpace[pt . Transpose[mb]] . mb . Transpose[qt]
    ] . qt,
    Norm[#] > 0 &
  ]
]

(* interpolation matrix P(S) *)
interp[s_] := Module[
  {mb = mask[s] . b, pt},
  For[pt = {}; i = 1, Length[pt] < Length[mb], i++,
    pt = extend[mb, pt, IdentityMatrix[Length[s]][[e[[i]]]]]
  ];
  Transpose[pt]
]

(* weight matrix W(S) for sample set S *)
weight[s_] := Module[
  {m = mask[s], p = interp[s]},
  b . p . Inverse[m . b . p] . m
]
```

Appendix II
Predictor Accuracy

We here investigate the accuracy of the bi-Lorenzian (L^2), edge (S_h), and radial (R) predictors for general (e.g. non-polynomial) functions f. Without loss of generality, consider predicting $f(0,0)$. If f is smooth or sufficiently densely sampled, we can approximate it well in the vicinity of the predicted sample by a second order Taylor series expansion g:

$$g_{x,y} = \sum_{i=0}^{2} \sum_{j=0}^{2} \frac{f^{(i,j)}(0,0)}{i!\,j!} x^i y^j$$

where $f^{(i,j)}$ denotes the i^{th} and j^{th} partial derivative with respect to x and y, respectively. We then have as prediction error $f(0,0) - p(0,0)$, where we express the prediction p as a weighted combination of samples $g_{x,y}$ according to the given predictor weights. For the bi-Lorenzian predictor, the error is:

$$f(0,0) - \big[2(g_{1,0} + g_{0,1} + g_{2,1} + g_{1,2}) - 4g_{1,1} - (g_{2,0} + g_{0,2} + g_{2,2})\big] = f^{(2,2)}(0,0)$$

Similarly, the edge predictor yields an error of $\frac{1}{2} f^{(2,2)}(0,0)$ and the radial predictor an error of $\frac{1}{4} f^{(2,2)}(0,0)$. Thus, in the limit the radial predictor is four times as accurate as the bi-Lorenzian.

Appendix III
Sensitivity to Noise

The sampled function often contains noise, either due to an imperfect acquisition process or due to errors stemming from limited precision and quantization. Assuming otherwise perfect prediction, we analyze how noise affects the accuracy of our predictors. We assume that each function value $f(i,j)$

involved in the prediction is perturbed by a small offset $\delta(i,j) \in [-\varepsilon,+\varepsilon]$ uniformly distributed in this interval, which well models effects such as integer quantization. The square error E^2 due to noise then reduces to:

$$E^2 = \big(\delta(0,0) - \sum_i \sum_j w(i,j)\delta(i,j)\big)^2$$

where $w(i,j)$ is the predictor weight of sample $f(i,j)$. We find the root mean square error via multiple integration of the $\{\delta(i,j)\}$ over the domain $[-\varepsilon,+\varepsilon]^n$ of the n-point stencil. For 3×3 neighborhoods, we obtain the RMS errors (expressed in multiples of ε) listed in Table II.

TABLE II.
NOISE-INDUCED ERRORS FOR SELECTED SPECTRAL PREDICTORS.

Stencil	Sample	Name	Error / ε
2×1	corner	constant	$\sqrt{\frac{2}{3}} \simeq 0.816$
3×3	face	radial	$\frac{\sqrt{3}}{2} \simeq 0.866$
3×1	edge	linear interpolation	$1 \simeq 1.000$
3×2	edge		$1 \simeq 1.000$
2×2	corner	Lorenzo	$\frac{2}{\sqrt{3}} \simeq 1.155$
3×1	corner	linear extrapolation	$\sqrt{2} \simeq 1.414$
3×3	edge	edge predictor	$\sqrt{3} \simeq 1.732$
3×2	corner		$2 \simeq 2.000$
3×3	corner	bi-Lorenzian	$2\sqrt{3} \simeq 3.464$

APPENDIX IV
AFFINE INVARIANCE

For any number $m > 0$ of known samples out of n, the spectral weights used to predict any given sample always add to one, which makes our predictor affine invariant. That is, the weight matrix $W = BP(MBP)^{-1}M$ satisfies $W1_n = 1_n$, where 1_n is a vector of n ones.

Lemma 4.1: The first n-entry column of the interpolation matrix P equals $[1 \quad 0 \quad \cdots \quad 0]^T$.

Proof: It is well known that the first eigenvector of the Laplacian $\mathcal{L}$ is $B_0 = \frac{1}{\sqrt{n}}1_n$, with a corresponding unique eigenvalue $\lambda_0 = 0$. Since any nonempty sample set is linearly dependent on MB_0, we always include it in the construction of the basis MBP by setting the first column of P to $[1 \quad 0 \quad \cdots \quad 0]^T$. ■

Theorem 4.2: For any nonempty sample set, $W1_n = 1_n$.

Proof: Let $v = [\sqrt{n} \quad 0 \quad \cdots \quad 0]^T$ be an m-vector. Due to Lemma 4.1, $Pv = [\sqrt{n} \quad 0 \quad \cdots \quad 0]^T$. Thus, $BPv = \sqrt{n}B_0 = 1_n$. Further note that $MBPv = 1_m$, and hence $v = (MBP)^{-1}1_m$. So $1_n = BPv = BP(MBP)^{-1}1_m = BP(MBP)^{-1}M1_n = W1_n$. ■

APPENDIX V
SELF-INTERPOLATION

We here prove the self-interpolating property of our spectral interpolator: given a set of samples S and a superset T interpolated from S, it matters not whether additional samples are interpolated from S only or from T. In other words, $W_T W_S = W_S$, where we have used subscripts to distinguish between weights computed from different sample sets. The proof begins with a lemma:

Lemma 5.1: For any $S \subseteq T$, there exists a matrix $X_{S,T}$ such that $BP_S = BP_T X_{S,T}$.

Proof: The columns of BP_S form a basis for the columns of M_S^T over the samples S, i.e. the columns of $M_S BP_S$ span the columns of $M_S M_S^T = I$. Since $M_T BP_T$ is a basis for $M_T M_T^T$, since $\text{ran}(M_T^T) \geq \text{ran}(M_S^T)$, and since basis functions are added in the same order to construct bases for S and T, we have that BP_T is also a basis for M_S^T over S. As a result, $\text{ran}(BP_T) \geq \text{ran}(BP_S)$, which implies that we can write BP_S as linear combinations of BP_T, i.e. there is a matrix $X_{S,T}$ such that $BP_S = BP_T X_{S,T}$. ■

Theorem 5.2: For any $S \subseteq T$, $W_T W_S = W_S$.

Proof: Let $W_S = BP_S(M_S BP_S)^{-1}M_S$ be the weight matrix for the sample set S. We have:

$$\begin{aligned} W_T W_S &= BP_T(M_T BP_T)^{-1}M_T BP_S(M_S BP_S)^{-1}M_S \\ &= BP_T(M_T BP_T)^{-1}M_T BP_T X_{S,T}(M_S BP_S)^{-1}M_S \\ &= BP_T X_{S,T}(M_S BP_S)^{-1}M_S \\ &= BP_S(M_S BP_S)^{-1}M_S \\ &= W_S \end{aligned}$$

■

ACKNOWLEDGEMENTS

This work was performed in part under the auspices of the U.S. Department of Energy by University of California, Lawrence Livermore National Laboratory under contract W-7405-Eng-48.

REFERENCES

[1] A. W. Cook, W. H. Cabot, P. L. Williams, B. J. Miller, B. R. de Supinski, R. K. Yates, and M. L. Welcome, "Tera-scalable algorithms for variable-density elliptic hydrodynamics with spectral accuracy," in *ACM/IEEE Supercomputing*, 2005, p. 60.

[2] G. W. Larson, "LogLuv encoding for full-gamut, high-dynamic range images," *Journal of Graphics Tools*, vol. 3, no. 1, pp. 15–31, 1998.

[3] D. F. Maune, *Digital Elevation Model Technologies and Applications: The DEM Users Manual*. American Society for Photogrammetry and Remote Sensing, 2001.

[4] M. J. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: Principles and standardization into JPEG-LS," *IEEE Transactions on Image Processing*, vol. 9, no. 8, pp. 1309–1324, 2000.

[5] G. Roelofs, *PNG: The Definitive Guide*. O'Reilly, 2003, http://www.libpng.org/pub/png/book/.

[6] C. Touma and C. Gotsman, "Triangle mesh compression," in *Graphics Interface*, 1998, pp. 26–34.

[7] V. Engelson, D. Fritzson, and P. Fritzson, "Lossless compression of high-volume numerical data from simulations," in *Data Compression Conference*, 2000, pp. 574–586.

[8] P. Ratanaworabhan, J. Ke, and M. Burtscher, "Fast lossless compression of scientific floating-point data," in *Data Compression Conference*, 2006, pp. 133–142.

[9] P. Lindstrom and M. Isenburg, "Fast and efficient compression of floating-point data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1245–1250, 2006.

[10] H. Kobayashi and L. R. Bahl, "Image data compression by predictive coding I: Prediction algorithms," *IBM Journal of Research and Development*, vol. 18, no. 2, pp. 164–171, 1974.

[11] E. H. Feria, "Linear predictive transform of monochrome images," *Image and Vision Computing*, vol. 5, no. 4, pp. 267–278, 1987.
[12] A. W. Paeth, "Image file compression made easy," in *Graphics Gems II*, J. Arvo, Ed. San Diego: Academic Press, 1991.
[13] L. Ibarria, P. Lindstrom, J. Rossignac, and A. Szymczak, "Out-of-core compression and decompression of large *n*-dimensional scalar fields," *Computer Graphics Forum*, vol. 22, no. 3, pp. 343–348, 2003.
[14] [Online]. Available: http://www.cc.gatech.edu/~lindstro/data/spectral/
[15] L. Ibarria, P. Lindstrom, and J. Rossignac, "Spectral predictors," *Data Compression Conference*, pp. 163–172, March 2007.
[16] C. Chrysafis and A. Ortega, "Efficient context-based entropy coding for lossy wavelet image compression," in *Data Compression Conference*, 1997, pp. 241–250.
[17] G. K. Wallace, "The JPEG still picture compression standard," *Communications of the ACM*, vol. 34, no. 4, pp. 30–44, 1991.
[18] M. Isenburg, I. Ivrissimtzis, S. Gumhold, and H.-P. Seidel, "Geometry prediction for high degree polygons," in *SCCG*, 2005, pp. 147–152.
[19] H. Zhang, "Discrete combinatorial Laplacian operators for digital geometry processing," in *SIAM Conference on Geometric Design*, 2004, pp. 575–592.
[20] G. Taubin, "A signal processing approach to fair surface design," in *ACM SIGGRAPH*, 1995, pp. 351–358.
[21] A. Mascarenhas, M. Isenburg, V. Pascucci, and J. Snoeyink, "Encoding volumetric grids for streaming isosurface extraction," in *3DPVT*, 2003, pp. 665–672.
[22] [Online]. Available: http://www.cc.gatech.edu/projects/large_models/ps.html