

Spectral Interpolation on 3×3 Stencils for Prediction and Compression

Lorenzo Ibarria

Georgia Institute of Technology, Atlanta, GA, USA

Email: redark@cc.gatech.edu

Peter Lindstrom

Lawrence Livermore National Laboratory, Livermore, CA, USA

Email: pl@llnl.gov

Jarek Rossignac

Georgia Institute of Technology, Atlanta, GA, USA

Email: 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 through several applications that predictive coding using our spectral predictor improves compression for various sources of high-precision data.

Index Terms—interpolation, prediction, compression, spectral basis, discrete cosine transform, irregular stencils

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 [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

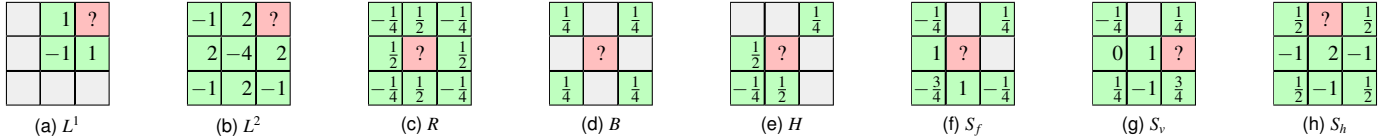


Figure 1. Weights for several spectral predictors: (a) Lorenzo, (b) bi-Lorenzian, (c) radial, (d) bilinear, (e) hybrid linear and radial, (f–h) full spectral.

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 scheme for compressing families of isocontours, and a set of appendices expanding and proving key properties of the spectral predictors. We prove that the sum of the predictor weights always add to unity, demonstrate the accuracy of several spectral predictors and their sensitivity to noise, provide Mathematica code to compute the spectral weights, and prove that the predictors are self-interpolating. We first cover the underlying theory of spectral prediction, and then provide example applications and results.

II. PRELIMINARIES

Before we derive our spectral predictor, we begin by considering the L^1 Lorenzo predictor [13] and its generalizations.

A. Extrapolating bi-Lorenzian predictor, L^2

Let f be a one-dimensional function regularly sampled at $\{\dots, f_{i-1}, f_i, f_{i+1}, \dots\}$, and let Δ^x be the finite difference $\Delta_i^x = f_i - f_{i-1}$. That is, Δ^x is an approximation of the differential $\frac{\partial f}{\partial x} dx$. Setting $\Delta_i^x = 0$, solving for f_i , and substituting L_i^1 for f_i , we have as 1D Lorenzo predictor $L_i^1 = f_{i-1}$. The Lorenzo predictor extends to 2D via composition of derivatives: $\Delta_{i,j}^{xy} = \Delta_{i,j}^x - \Delta_{i,j-1}^x = f_{i-1,j-1} - f_{i,j-1} - f_{i-1,j} + f_{i,j}$. As the sampling rate of f increases, Δ^{xy} approaches $\frac{\partial^2 f}{\partial x \partial y} dx dy$ in the limit. Setting $\Delta_{i,j}^{xy} = 0$, we can now express the 2D Lorenzo predictor as

$$L_{i,j}^1 = f_{i,j-1} + f_{i-1,j} - f_{i-1,j-1} \quad (1)$$

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 . Fig. 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

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

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

$$\begin{aligned} L_{i+1,j+1}^2 &= 2f_{i,j-1} + 2f_{i-1,j} + 2f_{i+1,j} + 2f_{i,j+1} \\ &\quad - 4f_{i,j} - f_{i-1,j-1} - f_{i+1,j-1} - f_{i-1,j+1} \end{aligned} \quad (2)$$

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 $x^2 y^2$. 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 $L_{i,j}^2 \neq L_{i,j-1}^1 + L_{i-1,j}^1 - L_{i-1,j-1}^1$. 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 Fig. 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.

B. 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 $\Delta_{i,j}^{xxyy} = 0$, 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

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

We use the term “radial” to describe this predictor because its weights are radially dependent on the distance to neighboring samples (Fig. 1(c)). The prediction $R_{i,j}$ is $2E - C$, where E is the mean of the four edge neighbors $\{f_{i\pm 1,j}, f_{i,j\pm 1}\}$ and C is the mean of the four corner neighbors $\{f_{i\pm 1,j\pm 1}\}$. $R_{i,j}$ 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 $x^2 y^2$ term, but typically yields better predictions due to the symmetric configuration of its neighborhood. Using Taylor expansion of f we show in Appendix II 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.

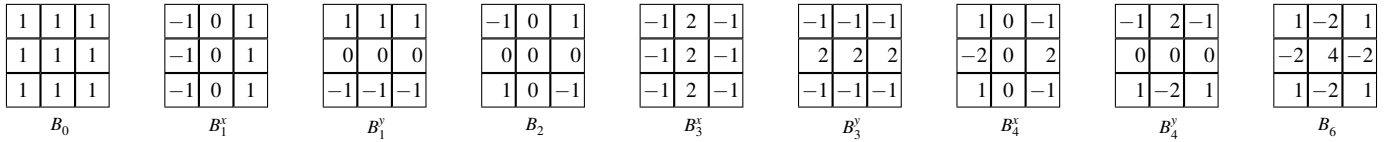


Figure 2. Basis functions for the 2D discrete cosine transform (not normalized).

III. 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 toward 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 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-A is in the spectral sense the optimal predictor (i.e. smoothest interpolant) for hypercube-like neighborhoods with one unknown sample.

A. Spectral derivation of L^2 and R

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

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

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

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

The cosine basis is shown un-normalized in Fig. 2. We unfold the 3×3 matrix into a single 9-dimensional vector $b = [f_{i-1,j-1} \ f_{i,j-1} \ f_{i+1,j-1} \ \cdots \ f_{i+1,j+1}]^T$ 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 $x = B^T b$, 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 (ω_x, ω_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])

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

where we consider the graph formed by the 3×3 neighborhood in isolation, with vertical and horizontal edges between adjacent samples. Here $\text{deg}(i)$ denotes the degree or number of neighbors of a sample i , e.g. $\text{deg}(i)$ is 2 for corner samples, 3 for edge samples, and 4 for face samples. As noted by Strang [20], the eigenbasis of the (un-normalized, symmetric, positive semidefinite) Laplacian coincides with the discrete cosine basis (the DCT-2 basis), and the eigenvalues $\{\lambda_i\}$ of \mathcal{L} are of the form

$$\lambda = 2(2 - \cos \omega_x - \cos \omega_y) = 4(\sin^2 \frac{\omega_x}{2} + \sin^2 \frac{\omega_y}{2}) \quad (5)$$

with $\omega = \frac{\pi}{3}k$, $k \in \{0, 1, 2\}$ for 3×3 neighborhoods. It follows that the eigenvalues of \mathcal{L} are $(0, 1, 1, 2, 3, 3, 4, 4, 6)$. Thus, larger eigenvalues λ correspond to higher frequencies ω , and from here on we will use the terms eigenvalue and frequency interchangeably. We will further 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 (Fig. 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,

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

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

	?	3/5
2/5		

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

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 (2) and (3) for corner and center predictions.

B. The general case: Irregular sample configurations

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 $b = [f_{i-1,j-1} \ f_{i,j-1} \ f_{i+1,j-1} \ 0 \ \dots \ 0]^T$.
- (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 Fig. 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.

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 $MBP = Mb$ 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 $W = BP(MBP)^{-1}M$. That is, row i of W gives the weights for interpolating an unknown sample b_i from its known neighbors: $b_i = \sum_j w_{ij}b_j$. Of course, if b_i is a known sample, only $w_{ii} = 1$ is nonzero.

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.

C. 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 Fig. 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.

IV. 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, inpainting, 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]).

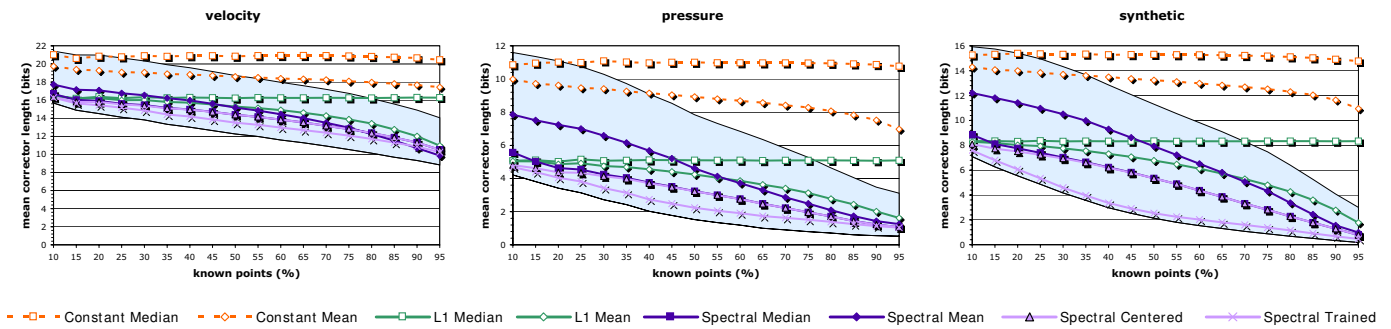


Figure 4. Predictor quality vs. average number of known points in the 3×3 neighborhood. The shaded area bounds the best and worst spectral prediction.

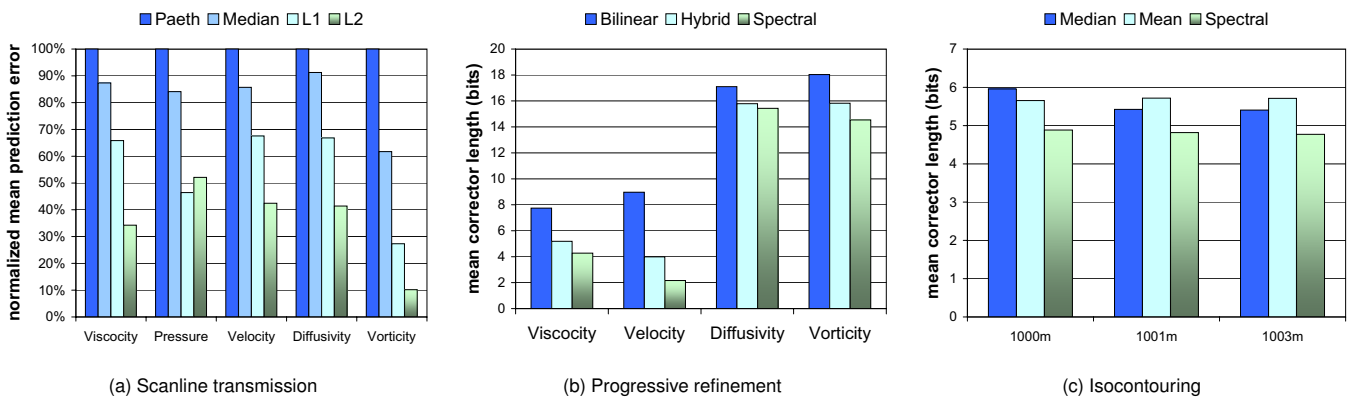


Figure 5. Prediction results for three different applications.

A. Scanline transmission

The most straightforward way 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 Lorenz predictor [13] used by Lindstrom and Isenbarg [9]. All except L^2 predict a sample from the same set of three neighbors (Fig. 1(a)).

In order to apply L^2 in a scanline traversal, two rows of previously coded samples must be maintained (Fig. 7(a)). To bootstrap the predictor, one may use lower-dimensional Lorenz prediction to first recover domain boundaries. Alternatively, one may use the spectral predictor for partially known neighborhoods described in Section III.

Fig. 5(a) shows the results of predicting multiple 2D slices of the single-precision floating-point scalar fields shown in Fig. 6 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 (see Appendix III). 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 Fig. 7(b), which results in three 3×3 neighborhood configurations from which samples are predicted (Fig. 7(b-d)).

We consider three predictors for the face sample (Fig. 7(c)): bilinear interpolation B of corner samples (Fig. 1(d)), spectral prediction S_f (Fig. 1(f)), and a hybrid predictor H (Fig. 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 (Figs. 1(g) and 1(h)).

Fig. 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 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 predicting the missing edge samples before 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.

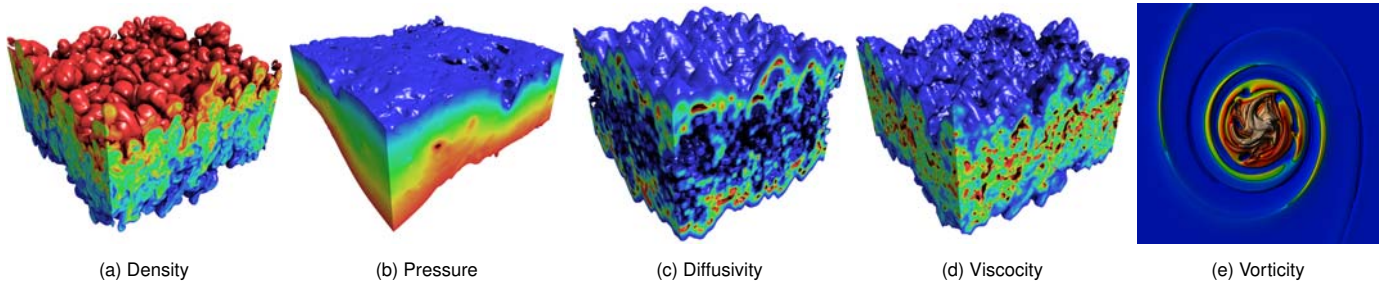


Figure 6. (a–d): Interval-volume renderings of several of the 3D scalar fields used in our experiments. (e): Close-up of the 1025×5000 2D vorticity field.

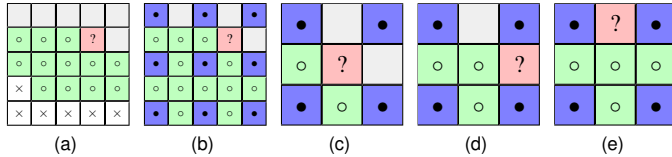


Figure 7. (a) L^2 footprint (circles) maintained during scanline traversal. (b) Coarse-resolution (solid) and fine-resolution (hollow) processed samples in a hierarchical traversal. Within each level of resolution, scanline traversal is used, resulting in three predictor stencils: (c) face, (d) vertical edge, and (e) horizontal edge sample.

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 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.

Although isocontouring of 3D volume data is a more common task than isocurve extraction from 2D fields, we focus here on the 2D case as it allows for straightforward application of our 3×3 predictor. Aside from the higher memory usage of the lookup tables needed by a 3D spectral predictor, the generalization from 2D to 3D is largely straightforward.

Our approach is based on the traversal, identification, and transmission of the minimal subset of samples that completely determine the isocontour requested by the client. We use an isocontour extraction method similar to *marching cubes* [22], but avoid traversing the entire 2D data set for each requested isocontour. Rather, we assume that a pre-computed set of seeds is available to the encoder from which it is possible to recover the isocontour [23]. The benefit of this seed set is that it allows visiting only those *cells* intersected by the isocontour. A cell is the rectangular region in the 2D grid between four neighboring

samples, which are pairwise joined by *links* (edges in the grid). We call the links intersected by the contour *sticks*, i.e. the two sample values of a stick bracket the given isovalue. As is common, the vertices of the isocontour are placed on the sticks by linearly interpolating the function values at the stick endpoints and solving for the position where this interpolant equals the isovalue.

The encoder and decoder perform the same traversal, hence we discuss them together. They both store in memory the surrounding sample values that determine the isocontour. Our implementation uses a quadtree as a sparse data structure for storing a mask of known samples and their values. This structure provides fast, adaptive storage, while maintaining spatial relationships between the samples.

1) *Compression Algorithm*: Our compressor encodes isocontours one component at a time. For each component, it begins with a seed stick and encodes its two sample values (A, B from Fig. 8). The encoder then follows the isocontour until it exits the domain of the data set or returns to the seed stick. During this traversal, we enter a cell through a stick and exit through another. While the isocontour never bifurcates, it is possible for a cell to have four sticks, in which case the contour passes through the same cell twice.

The two samples at the entrance stick are known by both encoder and decoder, however the other two samples of the cell (C and D in Fig. 8) may not be known by the decoder. This leads to the following three cases when entering a cell:

- (1) *Only the two samples from the entrance stick are known.* To determine the exit stick, we must first decide which of the samples C and D to encode first; it is possible that only one of them is needed to trace the isocontour. We use our spectral predictor to produce estimates C' and D' for the remaining samples. If $\overline{AC'}$ is a stick, we encode/decode C and continue to the next case below. Otherwise, we perform the same test for $\overline{BD'}$ and transmit D if the test passes. If neither link is estimated to be a stick, we encode C or D depending on which is closest to the intersection point between the isocontour and the entrance stick. This simple heuristic works well in practice, and often identifies the correct exit stick.
- (2) *Three samples are known, including the two samples A and B from the entrance stick.* Without loss of generality, we assume that C is the other known sample. If \overline{AC} is a stick, then it becomes the exit stick. Otherwise we encode/decode D , and determine through which link of \overline{BD} and \overline{CD} the isocontour exits the cell.

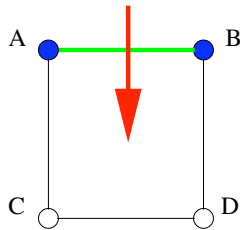


Figure 8. Example 2D cell. The entrance stick is shown in green, the already encoded samples in blue, and the traversed part of the isocontour in red.

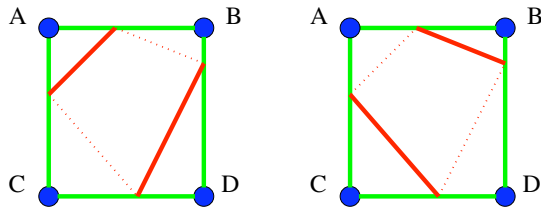


Figure 9. Ambiguous case in 2D isocontouring (*x-cells*).

- (3) *All four samples are known.* If in addition to the entrance stick the cell contains only one more stick, this becomes the exit stick (nothing needs to be encoded). Otherwise, all four links are sticks, and the contour passes through the cell twice. We call such cells *x-cells*. Without information on the behavior of the continuous function in the interior of the cell, there are two possible and equally valid interpretations of how to trace the isocontour (see Fig. 9). Several criteria have been proposed for resolving such ambiguities (e.g. [24], [25]), and we leave it up to the client to interpret such cells. However, in order to visit all samples needed by the isocontour, the encoder/decoder must agree on which cell to visit next. For simplicity, we use our heuristic above and exit through the link closest to the entrance vertex (intersection). Note that this choice affects only the *order* of samples encoded, i.e. it has no impact on which cells are traversed or which samples are needed to extract the isocontour.

To encode the value of a sample, we use spectral prediction and the residual coder from [9]. As discussed in Section III-C, we have freedom in positioning the neighborhood around the predicted sample. From the nine possible neighborhoods, we use as heuristic the one containing the largest number of known samples (though training the predictor, as in Section III-C, is also possible).

To bootstrap the decoder, we transmit with each component a start seed in the form of a stick. This stick is encoded as the location of its left-most, bottom-most sample, and a single bit indicating its orientation (horizontal or vertical). If no nearby samples are available, we use the isovalue w as prediction of the first sample f_0 . The other seed sample, f_1 , which together with f_0 must bracket w , is then predicted as $f_1 = 2w - f_0$.

It is important to point out that our approach never transmits the value of a sample more than once. If a sample was previously encoded with an isocontour, it will be available locally to the client for extraction of subsequent isocontours.

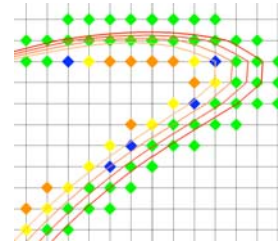


Figure 10. Close-up of a sequence of transmitted isocontours. The set of samples transmitted with each isocontour is shown in a separate color.

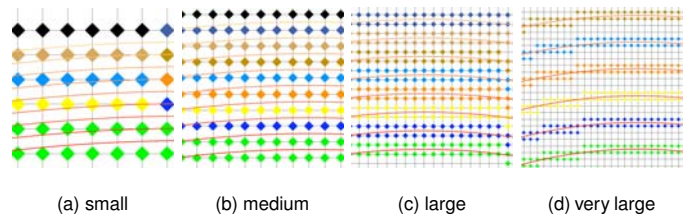


Figure 11. Four sets of isocontours from the synthetic circle data set. From left to right, the step between isovalues is doubled. Each isocontour and the samples they depend on are shown in a different color. As is evident, fewer samples are transmitted for more closely spaced isocontours.

2) *Results:* We have evaluated our isocontour compressor on three data sets: an isotropic distance map with circles as isocontours (Fig. 11), a $1,025 \times 5,000$ vorticity scalar field from a numerical hydrodynamics simulation by Miller et al. [26] (Fig. 6(e)), and the $16,385 \times 16,385$ Puget Sound 16-bit precision terrain surface [27]. The first two data sets are stored in single (32-bit) precision floating point.

Though it is common to measure isocontour compression in terms of the number of bits per generated isocontour *vertex* (bpv) transmitted, we will also evaluate our method in terms of number of bits per function *sample* (bps).

When our implicit technique is used to compress a single isocontour, it generally does not perform as well as methods that explicitly compress the isocontour representation directly. This is to be expected, as we encode information sufficient to extract a range of isocontours and to evaluate the function over a subset of the domain. For the 32-bit floating-point circle data set, our method compresses large isocontours to 17.13 bits per vertex, equivalent to 12.10 bits per sample. For this same isocontour, 4.2 neighbors are available on average for prediction. Because locally the isocontour is mostly “straight,” the neighborhood configuration is often such that the spectral predictor has to resort to 1D linear extrapolation, which is not a particularly powerful predictor.

Our method’s strength shows when we compress several nearby isocontours. Fig. 11 shows four sets of isocontours. As can be seen in Fig. 12, when compressing a set of isocontours that are far apart from each other, the end result is simply the compression of several isolated isocontours. As the distance between the different isocontours decreases, as seen in the examples “large,” “medium,” and “small,” the cost of encoding each new isocontour is reduced. The average number of neighbors used in the prediction for these three examples is 6.5, 7.3, and 7.4, respectively. The ratio between isocontour vertices and grid samples is 0.5, 0.7, and 1.2 respectively. The

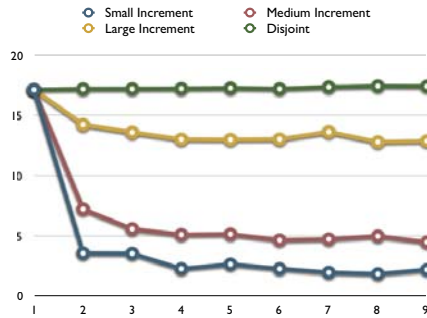


Figure 12. Compression in bits per vertex for each set of isocontours from Fig. 11. The horizontal axis corresponds to subsequent isocontours transmitted (from left to right). The graph shows that when isocontours are closely spaced together fewer samples need transmission and better prediction is possible.

“large” data set has almost the same sample/vertex ratio as the individual isocontour, but its prediction is improved by having nearby samples from previously transmitted isocontours. As long as the new isocontour is nearby already decompressed data, our technique offers improvement. The compression for the “very large,” “large,” “medium,” and “small” examples is 17.28, 13.64, 6.49 and 4.13 bits per vertex, respectively.

As a result of the way the samples 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 when isocontours are close the order does matter. To illustrate this point, we compressed the “medium” spaced set of isocontours in different orders. In the worst case, 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, resulting in compression similar to the “large” example. The second half of isocontours overlap those from the first half, and hence few additional samples are transmitted. Here 8.0 neighbors are available on average, which allows near perfect prediction at only 0.8 bits per vertex. The average cost across all isocontours is in this case 8.22 bpv. When instead the isocontours are transmitted in order of increasing isovalue, there is sufficient overlap for consistently good prediction, resulting in only 6.49 bits per vertex. Note, however, that the client may not be at liberty to choose this order as it may be dictated by the user.

We also evaluate our method on two differently spaced sets of nine isocontours each from the “vorticity” data set (Fig. 6(e)). The “large” isovalue range from this data set is twice as wide as the “medium” range. We achieve on average 16.6 bits per vertex on the “large” example, and 12.08 on the “medium.” By comparison, compressing a single isocontour requires 34.0 bpv. As in our other single-isocontour examples, the transmitted samples are often predicted poorly only via extrapolation along the isocontour, resulting in a cost similar to encoding each 1D stick intersection using a raw 32-bit floating-point representation.

As before, the initial overhead of our technique can be amortized over several isocontours, and again the transmission order matters (see Table I). We have observed in all cases that if the objective is to encode all the samples in a range, it is best to encode them sequentially (e.g. in order of increas-

TABLE I.
DEPENDENCE OF COMPRESSED FILE SIZE ON ISOCONTOUR
TRANSMISSION ORDER FOR THE MEDIUM-DENSITY 2D CIRCLE EXAMPLE.
ISOCONTOURS ARE NUMBERED BY INCREASING, EQUISPACED ISOVALUE.

Transmission Order	File Size
1,2,3,4,5,6,7,8,9	32537
1,3,5,7,9,2,4,6,8	41214
1,5,9,3,7,2,4,6,8	44599

ing isovalue) rather than hierarchically (e.g. by interleaving isocontours). In the sequential encoding we observed a 20% improvement over the hierarchical encoding. The improvement is entirely dependent on the accuracy of prediction since the same set of samples is eventually transmitted. In sequential transmission each point is predicted from a close-to-full stencil (7 to 8 neighbors), while hierarchical encoding is relegated to use sparsely populated stencils (4 neighbors on average) for half of the isocontours.

We end this section by reporting experiments of extracting isolines from the Puget Sound terrain surface. We first extracted an isocontour at 1000 m elevation and predicted all necessary samples, then incrementally transmitted missing values for isocontours at 1001 m and 1003 m, 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 (Fig. 5(c)).

V. CONCLUSIONS

We have described 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[Eigenvalues[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:

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

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 $\delta(i,j) \in [-\epsilon, +\epsilon]$ uniformly distributed in this interval, which well models effects such as integer quantization. The square error E^2 due to noise then reduces to:

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

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 $[-\epsilon, +\epsilon]^n$ 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	$\sqrt{\frac{2}{3}} \approx 0.816$
3×3	face	radial	$\frac{\sqrt{3}}{2} \approx 0.866$
3×1	edge	linear interpolation	1 ≈ 1.000
3×2	edge		1 ≈ 1.000
2×2	corner	Lorenzo	$\frac{2}{\sqrt{3}} \approx 1.155$
3×1	corner	linear extrapolation	$\sqrt{2} \approx 1.414$
3×3	edge	edge predictor	$\sqrt{3} \approx 1.732$
3×2	corner		2 ≈ 2.000
3×3	corner	bi-Lorenzian	$2\sqrt{3} \approx 3.464$

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 $[1 \ 0 \ \dots \ 0]^T$.

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 $[1 \ 0 \ \dots \ 0]^T$. ■

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

Proof: Let $v = [\sqrt{n} \ 0 \ \dots \ 0]^T$ be an m -vector. Due to Lemma 4.1, $Pv = [\sqrt{n} \ 0 \ \dots \ 0]^T$. 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, "Terascalable 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. Fritson, and P. Fritson, "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. Ivriissimtzis, 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. Strang, "The discrete cosine transform," *SIAM Review*, vol. 41, no. 1, pp. 135–147, 1999.
- [21] A. Mascarenhas, M. Isenburg, V. Pascucci, and J. Snoeyink, "Encoding volumetric grids for streaming isosurface extraction," in *3DPVT*, 2003, pp. 665–672.
- [22] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," in *Computer Graphics (Proceedings of SIGGRAPH 87)*, vol. 21, 1987, pp. 163–169.
- [23] M. van Kreveld, R. van Oostrum, C. Bajaj, V. Pascucci, and D. R. Schikore, "Contour trees and small seed sets for isosurface traversal," in *Proceedings of the 13th International Annual*

Symposium on Computational Geometry (SCG-97), 1997, pp. 212–220.

- [24] G. M. Nielson and B. Hamanna, “The asymptotic decider: resolving the ambiguity in marching cubes,” in *IEEE Visualization*, 1991, pp. 83–91.
- [25] C. Andujar, P. Brunet, A. Chica, I. Navazo, J. Rossignac, and A. Vinacua, “Optimal iso-surfaces,” *Computer-Aided Design and Applications*, vol. 1, no. 1–4, pp. 503–511, 2004.
- [26] P. Miller, P. Lindstrom, and A. Cook, “Visualizations of the dynamics of a vortical flow,” *Physics of Fluids*, vol. 15, no. 9, p. S13, 2003.
- [27] [Online]. Available: <http://www.cc.gatech.edu/projects/large-models/ps.html>

Lorenzo Ibarria was born in Barcelona, Spain, where he received his BS in computer science from the Polytechnic University of Catalonia. He did his graduate studies at the Georgia Institute of Technology, achieving a MS and PhD in computer science. His areas of interest are compression and computer graphics. Currently he is working at NVIDIA.

Peter Lindstrom was born in Sweden and has resided in the U.S. since 1990. He received a PhD in computer science from Georgia Institute of Technology in 2000, and holds BS degrees in computer science, mathematics, and physics from Elon University. He joined Lawrence Livermore National Laboratory in 2000, where he is a computer scientist and project leader. His primary research interests are in scientific visualization, data compression, mesh simplification, geometric and multiresolution modeling, and out-of-core techniques.

Dr. Lindstrom is a member of ACM and IEEE. He has served on 15 international program committees, and has published over 30 articles.

Jarek Rossignac was born in Poland and grew up in France. He received a PhD in electrical engineering from the University of Rochester, NY, in 1985. Between 1985 and 1996, he worked at the IBM T.J. Watson Research Center in Yorktown Heights, NY, where he served as Senior Manager and as Visualization Strategist. In 1996, he joined the Georgia Institute of Technology, Atlanta, GA, where he served as Director of the Gvu Center and is Full Professor in the College of Computing. His research focuses on the design, simplification, compression, and visualization of highly complex 3D shapes, structures, and animations.

Dr. Rossignac is a member of the ACM and a Fellow of the Eurographics Association. He authored 20 patents and over 100 articles, for which he received 13 Awards. He chaired the Solid Modeling Association; created the ACM Solid Modeling Conference series; chaired 20 conferences and program committees; and served on the Editorial Boards of 7 professional journals and on 52 Technical Program committees.