Efficient Local Statistical Analysis via Point-Wise Histograms in Tetrahedral Meshes and Curvilinear Grids

Bo Zhou, Yi-Jen Chiang *Member, IEEE* and Cong Wang

Abstract—Local histograms (i.e., point-wise histograms computed from local regions of mesh vertices) have been used in many data analysis and visualization applications. Previous methods for computing local histograms mainly work for regular or rectilinear grids only. In this paper, we develop theory and novel algorithms for computing local histograms in tetrahedral meshes and curvilinear grids. Our algorithms are theoretically sound and efficient, and work effectively and fast in practice. Our main focus is on scalar fields, but the algorithms also work for vector fields as a by-product with small, easy modifications. Our methods can benefit information theoretic and other distribution-driven analysis. The experiments demonstrate the efficacy of our new techniques, including a utility case study on tetrahedral vector field visualization.

Index Terms—Tetrahedral Meshes and Curvilinear Grids, Scalar Field Data, Vector Field Data, Geometry-Based Techniques, Mathematical Foundations for Visualization.

1 INTRODUCTION

One of the fundamental issues in data visualization and analysis is the computation of data statistics, be it global or local. In particular, histograms computed from local regions have been used in many data analysis and visualization applications. For example, they can be used for optimal viewpoint selection [48], for identifying material interfaces [49], [50], for transfer function design [29], and for tracking features in time-varying data [19]. Also, point-wise *local entropy* [46] (computed from local histograms) is used to guide streamline placement [52]; in the work [11], the computed point-wise entropy field is used to visually analyze the fluid pressure in flow simulations; in the *hixels* method [50], point-wise (local) histograms (called *hixels*) are used in the distribution-based algorithms to analyze and visualize scalar data. We envision that local histograms will play even more important roles in the coming years, since the ability to perform statistical analysis of the data distribution, and to quantify the uncertainty (or the information content) of the data is essential for guiding the data exploration process, especially when the size of data from simulations and data acquisition continues to grow exponentially.

Previous methods for computing local histograms mainly work for regular or rectilinear grids [11]; such methods are lacking for tetrahedral meshes or curvilinear grids. In this paper, we fill the gap by developing theory and novel algorithms for computing local histograms (specifically, point-wise histograms computed from local regions of mesh vertices) in such meshes/grids, which are widely used in computational fluid dynamics, shock physics (e.g. [34], [43], [47]), and so on.

After getting the local histogram for each mesh vertex v, we can compute various local statistic functions on v such as the *local entropy* (e.g. [11], [52]), *mutual information* between neighboring vertices (e.g. [50]), mean, standard deviation, etc., and store a few such values at v. We assume that at the end we only keep a few such values at each v, without keeping the whole local histogram bins, to save space. Below we use local entropy as a representative value to store.

Due to the irregular nature of the datasets, computing local histograms in tetrahedral meshes and curvilinear grids is quite challenging (Sec. 3). For tetrahedral scalar fields, we show how to apply *contour spectrum* [3] to obtain *accurate* results. Contour spectrum computes, for each cell C, the *accurate* function that gives the isosurface area inside C for each isovalue. However, it is mathematically proven that the distribution of isosurface area is *not* equal to the histogram distribution [15]; our method builds on the work [15] and is consistent with it.

We use a local neighborhood box for each vertex to compute its local histogram. Note that contour spectrum must be applied to a *whole* cell, and cannot work directly on a *partial* cell that is only partially inside the box (in general such partial cells cannot be avoided; see Sec. 3). To address this issue, we give a *clipping* approach (compute and triangulate the intersected regions, then apply contour spectrum) and prove its correctness, to obtain *provably accurate* results.

An even more important technical component of this paper is how to perform *sampling* correctly and efficiently in tetrahedral meshes and curvilinear grids. Since the contour spectrum method only works for *tetrahedral scalar fields*, sampling is needed for other cases: tetrahedral vector fields, and both scalar and vector fields of curvilinear grids. Even for tetrahedral scalar fields, clipping is too slow to be practical (see Sec. 4), and thus we need sampling to take care of partially intersected cells, together with contour spectrum on wholly contained cells.

A major issue in sampling is how to assign sampling weights. It is shown that for histograms computed from sampling, each sample should be weighted by the volume of its Voronoi cell [15]. While such volume weight is trivial ($1/N$ of the domain volume for N samples) in the regular grids considered [15], it is much more complex in our cases (Sec. 3) and thus too slow to compute (Sec. 4). We propose a novel approach based on the *barycentric*

• *The authors are with CSE Dept., Tandon School of Engineering, New York University, Brooklyn, NY, USA.*
Email: bz387@nyu.edu; chiang@nyu.edu; cw1068@nyu.edu.

dual (defined in the book [6]), for both tetrahedral meshes and curvilinear grids. We also establish the correctness by proving the *convergence* for *both* scalar and vector fields. Moreover, we explore the *geometric properties* of barycentric duals, so that the volume weights of samples can be obtained much more easily.

In addition to developing new theory, we also devise novel algorithms for computing local histograms/statistic functions in tetrahedral meshes and curvilinear grids, called *cell sampling with sweeping*, to achieve both time- and space-efficiency. They can benefit information theoretic and other distribution-driven analysis in a scalable manner. The experiments demonstrate the efficacy of our new techniques, including a utility case study on tetrahedral vector field visualization.

We can summarize the contributions of this paper as follows.

- We give novel sampling approaches based on *barycentric dual* for both tetrahedral meshes and curvilinear grids. They allow fast computation and converge quickly.
- We develop new theory for computing local histograms/statistic functions (e.g. entropy) in tetrahedral meshes and curvilinear grids, including geometric properties of barycentric duals and proof of convergence, as well as contour spectrum and clipping that produce *provably accurate* results for tetrahedral scalar fields.
- We devise novel *cell sampling with sweeping* algorithms for computing local histograms/statistic functions in tetrahedral and curvilinear scalar fields. They are theoretically sound and efficient, and work effectively and fast in practice.
- As a by-product, our scalar-field algorithms above can be extended for vector fields with small, easy modifications. We also provide a utility case study on tetrahedral vector field visualization.

Limitations: Our curvilinear-grid method requires that each cell be convex and the vertices of each face of a cell be co-planar. These conditions are typically true in practice, however. (E.g., in VTK curvilinear grids are usually represented as structured grids,[1] where each cell is a vtkHexahedron, a polyhedron with the vertices of each face co-planar. Also, VTK uses isoparametric interpolations for the cell interior, which require the hexahedron to be convex [1].)

2 Previous Work

Histograms are a common tool to display data distributions, and have been widely used for user interaction in transfer function design [29], [30], [31], [40], [44]. Methods exploiting the continuity in the gradient-intensity domain via 2D histograms for approximating the spatial continuity in the datasets include the work [25], [31], [40], [51]; one such approach [22] is based on multilevel segmentation.

Shannon's entropy [46] in information theory is a measure of the amount of information or uncertainty in data. It has been widely used in image processing [20] and computer vision [38]. Such entropy is typically computed with histograms. For data analysis and visualization, the concept of entropy/histograms has been actively exploited. This includes local entropy and conditional entropy for streamline generation [52], viewpoint entropy for optimal viewpoint selection [48], and isosurface similarity maps for isovalue selection [7]. Local histograms have also been used to identify material interfaces [49], [50] and to track features in time-varying data [19]. An efficient method for local statistical analysis is given using integral histograms with discrete wavelet transform [26]. For other information-theoretic results in this active area, see the recent book [12] for an excellent survey.

The relationship between histograms and isosurface statistics for regular sampling lattices was explored in a series of papers [9], [15], [41]: first, the fundamental relationships between statistics, geometry and algorithmic performance were identified [9]; next, some errors of the first paper [9] were corrected and the Federer's Co-Area Formula [16] was first introduced in this context [41]. Finally, the latest theory in this area was developed [15], by integrating the roles of statistics, geometry, algorithmic performance and measure theory. We also use the Federer's Co-Area Formula, and our methods build on the theory in the work [15].

In *continuous scatterplot* [2], when the input domain is 3D, the output with domain dimension 1 is a *continuous histogram*, and the method is equivalent to the one [15] mentioned above. In this sense our (continuous) B-spline function obtained from contour spectrum plays a similar role as the *density function* to construct the continuous scatterplot via integration, albeit using an interesting alternative. For the output with domain dimension 2 (continuous 2D scatterplot), the authors employed a tetrahedral cell projection for *volume-rendering* the continuous 2D scatterplot [2]. However, similar to contour spectrum, such a method only works for scalar fields (since volume rendering is restricted to scalar fields) and only applies to a *whole* cell, and cannot deal with partially intersected cells (unless using our *clipping*, which is too slow to be practical (Sec. 4)). The concept of fractal dimension of isosurfaces was defined and explored [24]. The barycentric dual was defined in the book [6]; it was used in FIT (finite integration technique) [14] and FEM (finite element method) [8] in computational electromagnetics.

GPU-based parallelization of entropy/histogram computation is well studied [17], [37], [42]. Also, the wide use of mutual information has motivated research on its GPU-based parallel computing [28], [45]. Almost all these methods work for regular grids only; methods for rectilinear grids were given in the paper [11]. Here we present novel approaches for computing local histograms/entropy in tetrahedral and curvilinear datasets.

3 Our Approach

We use a *local neighborhood box* for each vertex to compute its local histogram. Note that local statistical analysis is mainly used to quantify the amount of information between different vertices, to see whose region is more salient (i.e., has more features/information). Thus it is important that the box is of the *same size* for all vertices — so that we get the amount of information *per unit region* (rather than using varying sizes according to the local mesh resolution, which is incorrect). Therefore, for a given parameter t, at *each* vertex v we use a *neighborhood box of size* t, defined as a 3D cube centered at v with side length $2t$ in each dimension, and construct a histogram within the box. We do the same for v *at the mesh boundary*, but the portion of the box outside the mesh gets *no* contribution. We then use such local histogram to compute and store a few statistical values such as the *local entropy* (defined next) at v. In this way, we use the *same* value t for each vertex v. We discuss how to choose a reasonable value t in the Appendix (Supplementary Materials).

1. Though for special needs VTK also supports non-linear curvilinear grids by vtkQuadraticHexahedron [1].

We remark that a natural alternative to a neighborhood box is a neighborhood *ball* of radius t around each vertex v. However, our provably accurate approach, clipping, only works for neighborhood boxes but not neighborhood balls — for the former, the intersection between a tetrahedral cell and a neighborhood box is a "flat" shape and can be tetrahedralized to apply contour spectrum; for the latter, the intersection is a *curved* shape and there is no way to partition it into tetrahedra, and thus contour spectrum cannot be applied. Since we need clipping to obtain the ground truth to evaluate the accuracy of our final algorithm *cell sampling with sweeping*, we stick to neighborhood boxes throughout this paper. On a separate note, our final algorithm can be slightly modified to work for neighborhood balls (see the Remarks item 2 at the end of Sec. 3.1.C), although we do not know how to obtain the ground truth to evaluate its accuracy.

Formally, for a discrete random variable X with possible values $\mathscr{X} = \{x_1, x_2, ..., x_n\}$ and the probability mass function $p(x)$, Shannon's entropy [46] of X is defined as

$$H(X) = -\sum_{x_i \in \mathscr{X}} p(x_i) \log_2 p(x_i). \quad (1)$$

The entropy is a measure of the average uncertainty in X; larger entropy means more uncertainty, i.e., more information. The entropy in Eq. (1) is commonly computed by a histogram (e.g. [50], [52]). We can also compute/estimate standard deviation (SD) from a histogram [32]. At each vertex v we compute a local histogram as above, which is then used to compute the local entropy/SD at v.

3.1 Tetrahedral Meshes

For ease of exposition, in this section we focus on tetrahedral scalar fields. We first discuss our sampling approach, with the main technique in Sec. 3.1.A. We then discuss how to apply contour spectrum (including the clipping approach) in Sec. 3.1.B, followed by our novel efficient algorithm in Sec. 3.1.C. An easy extension to vector fields as a by-product is given in Sec. 3.3.

Intuitively, one would generate $k \times k \times k$ axis-aligned samples regularly (evenly spaced) inside the neighborhood box, and for each sample p, interpolate to get its data value p_v; p is then assigned to the histogram bin whose value range contains p_v, and a weight of $1/k^3$ of the box volume is added to this bin. We call such method *box sampling*. However, in order to perform interpolation, we need to locate the cell containing p for each p. Such *batched cell location* queries are very expensive even after decent accelerations with an octree (Sec. 4).

To overcome this difficulty, we use the following *cell sampling* idea: For each cell C intersected by the box N, generate sample points in C and assign them to the corresponding histogram bins only when they fall inside N. In this way, cell location queries are completely avoided, and filtering the sample points against N is easy since N is an axis-aligned cube. However, these sample points are no longer evenly spaced inside N and we need to assign each sample point p a suitable *volume weight* w to *accurately* account for its contribution.

Another base-line method is *Monte Carlo sampling*: *randomly* and uniformly generate n_s sample points from a domain, each with weight $1/n_s$ of the domain volume. Doing it for box sampling would have the same problem of cell location and is too slow. Doing it for cell sampling is more feasible, but it typically converges very slowly (Sec. 4).

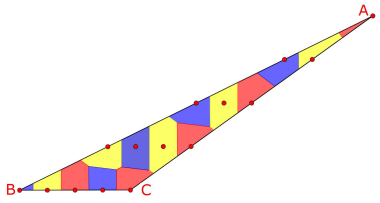


Fig. 1: 2D example of the Voronoi diagram on barycentric sample points where the Voronoi cells are irregular and their areas are difficult to compute.

3.1.A. Sampling and Weighting with Barycentric Dual

To get regular samples for each tetrahedral cell, we use *barycentric sampling* [39]. Let v_0, v_1, v_2, v_3 be the cell vertices. We regularly sample the cell along the 3 barycentric axes $(v_1 - v_0)$, $(v_2 - v_0)$, $(v_3 - v_0)$ [39], so that each barycentric axis/edge is evenly subdivided into k segments with $k+1$ samples (e.g., the red points in Fig. 4a for $k = 4$).

As mentioned, it is shown that for histograms computed from sampling, each sample should be weighted by the *volume of its Voronoi cell* [15]. Clearly it is too expensive to compute the Voronoi diagram on all sample points. Moreover, although barycentric samples are "regular", their Voronoi-cell volumes can be quite irregular and difficult to compute (see Fig. 1 for a 2D example).

Rather, we propose to use *barycentric dual* (defined in the book [6]) to assign the volume weights to the samples, where we explore the nice *geometric properties* of barycentric dual to get the volume weights easily. We establish the correctness by proving the *convergence* of our method.

Barycentric Subdivision (BCS)

To start, we define the *barycenter* of a polygon/polytope as the arithmetic mean (i.e., the "average") position of all the vertices of that polygon/polytope (it is also called the *centroid* of the polygon/polytope).

In geometry, the *barycentric subdivision (BCS)* is a standard way of dividing an arbitrary convex polygon/polyhedron into triangles/tetrahedra, or, in general, a convex polytope into simplices of the same dimension, by connecting the barycenters of their faces in a specific way. Our definition of BCS is consistent with that in the books [21], [36]. Note that we only apply BCS *once*.

The BCS of a triangle S divides it into 6 triangles; each part has one vertex at the barycenter of S, another one at the barycenter (midpoint) of some edge e of S, and the last one at a vertex v of S that is also an endpoint of e. For example, in Fig. 2a, O is the barycenter of triangle ABC, and P, Q, R are the barycenters (midpoints) of edges $\overline{AB}, \overline{BC}, \overline{CA}$, and triangle OPA is one of the 6 resulting triangles. (There are 3 choices for the edge e, and 2 choices for v. Overall there are $3 \cdot 2 = 6$ triangles in the BCS.)

The BCS of a tetrahedron S divides it into 24 tetrahedra; each part has one vertex at the barycenter of S, one at the barycenter of some face f of S, one at the barycenter (midpoint) of some edge e of S that is also an edge of f, and the last one at some vertex v of S that is also an endpoint of e. For example, in Fig. 2b, O is the barycenter of tetrahedron $ABCD$, Q is the barycenter of face ABC, and P is the midpoint of edge $\overline{AB}$; tetrahedron $OQPA$ is one of the 24 resulting tetrahedra. (There are 4 choices for the face f, 3 choices for the edge e, and 2 choices for v. Overall there are $4 \cdot 3 \cdot 2 = 24$ tetrahedra.)

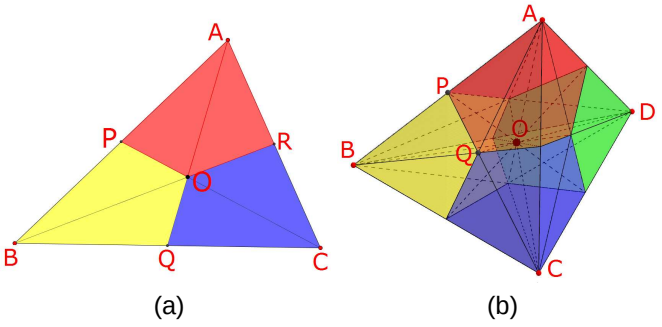


Fig. 2: Examples of barycentric subdivisions (BCS) for simplices: (a) 2D case; (b) 3D case.

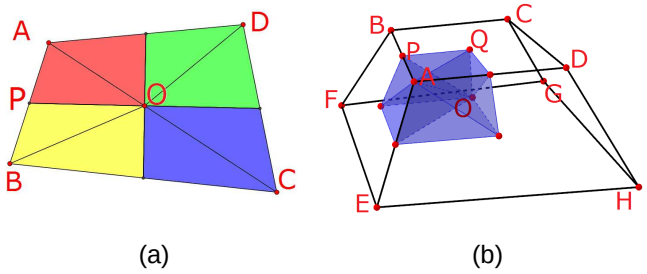


Fig. 3: Examples of barycentric subdivisions (BCS) for convex polytopes: (a) a 2D example; (b) a 3D example.

The above definition extends to the BCS of a 3D *convex polytope* into a number of 3D simplices. For example, in Fig. 3a (a 2D case), O is the barycenter of the polygon $ABCD$, P is the barycenter (midpoint) of edge $\overline{AB}$, and triangle OPA is one of the resulting triangles of the BCS. In Fig. 3b (a 3D case), O is the barycenter of the polytope $ABCDEFGH$, Q is the barycenter of the face $ABCD$, P is the barycenter (midpoint) of edge $\overline{AB}$, and tetrahedron $OQPA$ is one of the resulting tetrahedra of the BCS.

Weighting by Barycentric Dual (BD)

Now let us consider a tetrahedral cell C; recall that we use barycentric sampling to get regular samples in C. First, we *cut* C by planes that are parallel to the original faces of C and are going through the sample points; see Fig. 4a for a 2D example. Then, for each resulting convex polytope (each a triangle in Fig. 4a) we *perform barycentric subdivision (BCS)* to get the final simplices; see Fig. 4b. For each sample point p_i, we collect all final simplices (resulting from BCS) that are incident on p_i; the union of such final simplices is called the *cell* of the *barycentric dual (BD)* centered at p_i. For example, in Fig. 4b, the red points are the barycentric sample points and their BD cells are colored with red, blue or yellow. In our scheme, we assign each sample point p_i a volume weight that is the volume of the BD cell of p_i.

Geometric Properties of Barycentric Dual

From the above definitions, the barycentric dual (BD) of our barycentric sample points are very easy to compute. Moreover, we will explore some nice geometric properties so that the volume weight, i.e., the volume of the BD cell of each sample point p_i, is extremely simple to obtain — at the end we *do not need to compute BCS or BD.*

Let us consider the 2D case first.

Property 1: Referring to Fig. 2a, each of the 6 resulting triangles of the BCS has the same area, namely 1/6 of the area of triangle ABC.

Proof: See Appendix I (Supplementary Materials).

Similarly, one can show that the same property holds in 3D, and thus each of the 24 tetrahedra resulting from the BCS (see Fig. 2b) has 1/24 of the volume of the tetrahedron $ABCD$.

Now consider the volume of the barycentric-dual cells. In 2D, for example, from Fig. 4a we see that each edge (barycentric axis) of triangle ABC is subdivided into 4 segments in the barycentric sampling and there are $16 = 4^2$ congruent triangles after cutting, and in Fig. 4b each such triangle is further subdivided (by BCS) into 6 triangles/simplices of the same area (Property 1), i.e., each final simplex has area $(1/6)\cdot(1/4^2)$ of the area of triangle ABC. Now we see that there are 3 types of sample points: **(1)** at the vertices (A, B or C), **(2)** on the edges of triangle ABC, and **(3)** in the interior of triangle ABC. For **(1)**, its BD cell consists of 2 final simplices; for **(2)** its BD cell consists of 6 final simplices; and for **(3)** its BD cell consists of 12 final simplices (see Fig. 4b). Thus the area of each type is $1/3$, 1 and 2 times $(1/4^2)$ of the area of triangle ABC, respectively. In general, if each edge of triangle ABC is subdivided into k segments in the barycentric sampling then we only need to replace 4^2 by k^2 (since now there are k^2 congruent triangles), and all the rest remains the same.

We can extend the above property to a 3D tetrahedral cell. Recall that each barycentric axis/edge is subdivided into k segments. There are 4 types of sample points: **(a)** at the vertices, **(b)** on the edges, **(c)** on the faces, and **(d)** in the interior. Deriving their weights is trickier, but after derivation the actual computation is extremely simple.

For a tetrahedral cell C, recall that the first step to obtain the BD cells for the barycentric sample points is to *cut* C by planes that are parallel to the original faces of C and are going through the sample points. This results in smaller tetrahedra that are *similar to* (i.e., of the *same shape* as) the original cell C; the remaining parts are all octahedra. See Fig. 5 for examples. (Here the resulting convex polytopes are tetrahedra and octahedra, as opposed to just triangles in the 2D case (Fig. 4a).) Let V be the volume of the original cell. Then each (blue) tetrahedron has volume $(1/k^3)V$ and each (red) octahedron has volume $(4/k^3)V$, as seen in Fig. 5a for $k = 2$: each of the 4 blue tetrahedra has volume $(1/8)V$; the remaining volume, $(4/8)V$, is the volume of one red octahedron. Thus *a red octahedron has 4 times the volume of a blue tetrahedron*. This is true for general k.

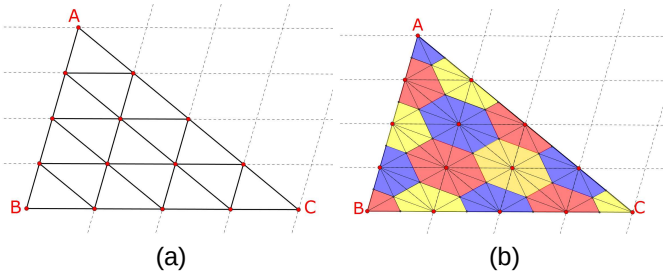


Fig. 4: The barycentric dual of sample points: (a) *cutting* a cell (triangle ABC) by lines parallel to the cell edges and going through the barycentric sampling points, where the sample points are shown in red; (b) the barycentric dual (BD), where the BD cells of the sample points are colored with red, blue or yellow.

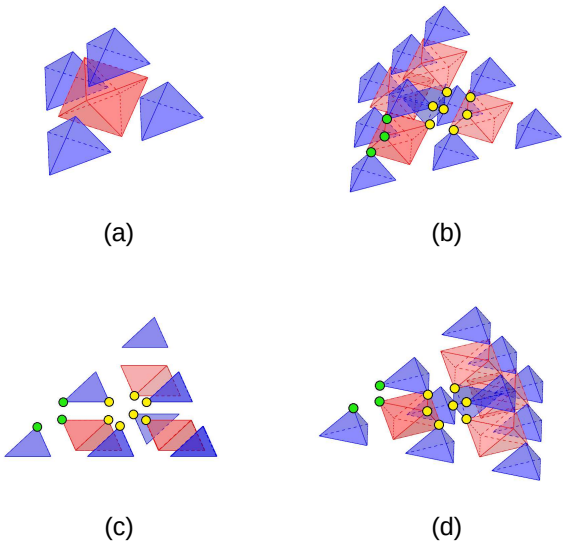


Fig. 5: *Cutting* a tetrahedral cell C by planes that are parallel to the original faces of C and are going through the sample points, where each barycentric axis/edge is subdivided into k segments by the sample points. (a) An example when $k = 2$. (b)-(d) An example when $k = 3$, with the same configuration seen from different viewpoints, where green points denote a sample point on an edge of C, and yellow points denote a sample point on a face of C (this is easiest to see in (c)).

To obtain the weights for our sample points, we consider the BCS on such blue/red polytopes and derive the volume weights of their vertices.

Property 2: Each of the 4 vertices of a blue tetrahedron has the *same* volume weight, i.e., 1/4 of the volume of the blue tetrahedron.

Proof: See Appendix I (Supplemental Materials).

Property 3: Each of the 6 vertices of a red octahedron has the *same* volume weight, i.e., 1/6 of the volume of the red octahedron.

Proof: See Appendix I (Supplementary Materials).

Now we can derive the volume weights for the 4 types of sample points. Let V be the volume of the original tetrahedral cell. Recall that each blue tetrahedron has volume $\frac{1}{k^3}V$ (where each vertex gets 1/4 of it (Property 2)) and each red octahedron has volume $\frac{4}{k^3}V$ (where each vertex gets 1/6 of it (Property 3)). For **(a)** sample points at the cell vertices, they are weighted by $\frac{1}{4}\cdot\frac{1}{k^3}V$. For **(b)** sample points on the edges (see green points in Fig. 5(b)-(d)), their weights are $2\cdot(\frac{1}{4}\cdot\frac{1}{k^3}V)+(\frac{1}{6}\cdot\frac{4}{k^3}V)=\frac{7}{6k^3}V$, contributed by 2 blue tetrahedra and 1 red octahedron. For **(c)** sample points on the faces (see yellow points in Fig. 5(b)-(d)), their weights are $4\cdot(\frac{1}{4}\cdot\frac{1}{k^3}V)+3\cdot(\frac{1}{6}\cdot\frac{4}{k^3}V)=\frac{3}{k^3}V$, contributed by 4 blue tetrahedra and 3 red octahedra. For **(d)** interior sample points, the weights are twice as the ones on the faces, which are simply $\frac{6}{k^3}V$. We now summarize the final results on sampling weights.

Theorem A: *Let V be the volume of the original tetrahedral cell, and each barycentric axis is subdivided evenly into k segments by $k+1$ samples. Then the sample points of the 4 types — **(a)** at the cell vertices, **(b)** on the edges, **(c)** on the faces, and **(d)** in the interior — have weights $\frac{1}{4k^3}V$, $\frac{7}{6k^3}V$, $\frac{3}{k^3}V$, and $\frac{6}{k^3}V$ respectively.*

Note that after computing the cell volume V, we can **directly assign** the volume weights of the sample points, **without computing** the barycentric subdivision (BCS) or barycentric dual (BD).

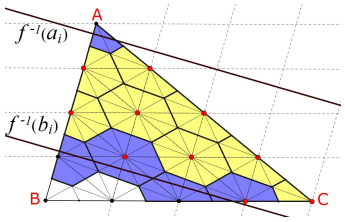


Fig. 6: Proof of convergence of histograms computed by our barycentric-dual approach. The lines labeled $f^{-1}(a_i)$ and $f^{-1}(b_i)$ are the level sets at scalar values a_i and b_i that intersect this triangle cell.

Proof of Convergence

It is proven that in a regular grid, the histogram H_N converges to π_f as the number N of sample points tends to infinity [15]:

$$\lim_{N\to\infty} H_N(i) = \pi_f(i), \tag{2}$$

where $H_N(i)$ is the histogram at bin i with value range a_i to b_i, and $\pi_f(i)$ is the level-set measure that measures the size of the level set from scalar value a_i to b_i. We will give a similar proof that, in a tetrahedral mesh, our histogram computed using BD of sample points also converges to the level-set measure.

Since we assume linear interpolation in each mesh cell, the isosurfaces in the mesh cell are parallel to each other, and the interval region between two isosurfaces in the cell is continuous. We find the lower bound and the upper bound of $\pi_f(i)$. Our histogram $H_N(i)$ collects the BD cells which have their centers (i.e., sample points) inside the interval region. Therefore the lower bound $L_N(i)$ can be the BD cells that are fully contained in the level set region (the yellow cells in Fig. 6). The upper bound $U_N(i)$ comes from the BD cells that intersect the level set region (the yellow cells together with the blue cells in Fig. 6). Therefore, we have

$$L_N(i) \le H_N(i) \le U_N(i). \tag{3}$$

As the number N of sample points tends to infinity, it is clear that either $L_N(i)$ or $U_N(i)$ computes the size of the interval region:

$$\lim_{N\to\infty} L_N(i) = \lim_{N\to\infty} U_N(i) = \pi_f(i). \tag{4}$$

By applying the Squeeze Theorem, as N goes to infinity, $H_N(i)$ is squeezed to $\pi_f(i)$, which proves Eq. (2) as desired.

3.1.B. Applying Contour Spectrum

Discrete sampling presented so far is only an approximation since in practice we can only use a finite number of sample points. In tetrahedral scalar fields, for a cell C that lies *entirely inside* the neighborhood box N, we can apply *contour spectrum* [3] to improve both the accuracy and efficiency. Let $f(\mathbf{x})$ be the scalar-field value at location $\mathbf{x}$. Contour spectrum gives a B-spline function $g(h) = A(f^{-1}(h))$, which maps each isovalue h to the *accurate* area of its isosurface in C [3]. Then according to the partition of the histogram bins, we integrate $g(h)$ on each histogram bin span $[a_i, b_i]$ with respect to the gradient:

$$V_i = \int_{a_i \le f(\mathbf{x}) \le b_i} 1 \mathrm{d}V = \int_{a_i}^{b_i} \int_{f^{-1}(h)} |\nabla f(\mathbf{x})|^{-1} \mathrm{d}S\mathrm{d}h. \tag{5}$$

Formula (5) is proven and confirmed ([15], [41]) using Federer's Co-Area Formula [16]. The integration result is the

Hausdorff measure [16], i.e. the volume which is contributed to the histogram bin i. Furthermore, using contour spectrum, we assume linear interpolation of $f(\mathbf{x})$ with each cell C. So we can compute the gradient magnitude as a constant G for C:

$$G = |\nabla f(\mathbf{x})| = \frac{f(\mathbf{v_4}) - f(\mathbf{v_1})}{(\mathbf{v_4} - \mathbf{v_1}) \cdot \vec{g}}, \tag{6}$$

where $\vec{g}$ is a unit vector in the gradient direction, $\mathbf{v_1}$ and $\mathbf{v_4}$ are vertices of C with the minimum and maximum scalar-field values.

Thus we can simplify formula (5) as

$$V_i = \int_{a_i}^{b_i} \frac{g(h)}{G} \mathrm{d}h. \tag{7}$$

For a special case where the cell has no span, i.e. $f(\mathbf{v_4}) = f(\mathbf{v_1})$ or $G = 0$, we just compute the volume of C and contribute it to the corresponding histogram bin. This is consistent with the paper [15].

The authors of contour spectrum [3] confirmed that the formula given in the original paper [3] was slightly imprecise for the 3D case. We give an accurate formula and also simplify it slightly, from a piece-wise B-spline function over 5 intervals to over 3 intervals (see Appendix II in Supplementary Materials).

Clipping and its Correctness

Recall that contour spectrum can only apply to a *whole* cell. For a cell C that is partially intersected by the box N, we can use a *clipping* approach: Compute and triangulate the intersected regions, and then apply contour spectrum on each resulting tetrahedron. Note that triangulating mesh cells is typically *not* a valid method, since different triangulations can lead to different results [10]. However, we can show that *clipping* does not have this problem — different triangulations will always lead to the *same* (and thus correct) result.

Proof: Since inside C we assume linear interpolation from its 4 vertices, any newly created vertex from intersecting C and N gets its scalar value from the same linear interpolation scheme. Also, under the same linear field of C, the triangulated regions all have the same gradient direction $\vec{g}$ no matter how we triangulate them. Moreover, the isosurfaces considered in contour spectrum are all perpendicular to $\vec{g}$, and thus the results accumulated to the histogram bins are always the same regardless of different triangulations. Thus clipping is a correct and *provably accurate* method.

3.1.C. Efficient Algorithm

As will be seen in Sec. 4, clipping is extremely slow, and thus we want to apply sampling on partially intersected cells instead. For a neighborhood box N and a cell C, if all 4 vertices of C lie inside N then C is *fully contained* in N and vice versa; this is an easy case (and we will apply contour spectrum). However, to test whether C is *partially intersected* by N precisely is quite complex and expensive. For example, in Fig. 7, the red box and the blue striped triangle cell intersect, but none of their corners/vertices lie in the interior of each other (similarly for 3D).

Instead of performing precise tests, we use a cheap but *conservative* test: we replace C by its axis-aligned bounding box $B(C)$ and test against N: if $B(C)$ intersects N and C is not fully contained in N, then C is *potentially partially intersected* by N. Since $B(C)$ and N are both axis-aligned, their intersection test is easy, and we will never miss any real intersections. It is possible that $B(C)$ intersects N but C does not, then we will generate sample points from C but they are all filtered out by N in the sampling step, which still gives the correct result.

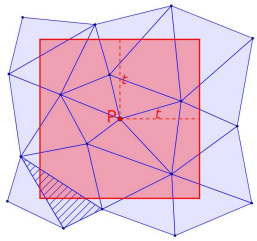


Fig. 7: A 2D example where the neighborhood box (shown in red) and a mesh cell (the blue striped triangle) intersect but none of their corners/vertices lie in the interior of each other.

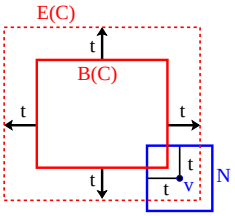


Fig. 8: Key idea of our KD-tree queries: $B(C)$ intersects N if and only if $E(C)$ contains v, where N is the box of size t around v.

If we take each N and consider the candidate cells C for their contributions to N, then sample points from the same cell C can be generated again and again for different vertices v (and their boxes N), which is inefficient. Rather, we use the *cell sampling* method: for each cell C, we apply contour spectrum on C *once*, and distribute it to the boxes N *fully containing* C; we also generate the sample points from C only *once*, and distribute them to the boxes N that *potentially partially intersect* C. Now the task is: Given a cell C, how do we find the candidate vertices v whose boxes N potentially intersect C?

Consider enlarging the bounding box $B(C)$ by pushing each face outward in the normal direction with a distance t; call the resulting box *enlarged box* $E(C)$ (Fig. 8). The key idea is this: $B(C)$ *intersects* N *if and only if* $E(C)$ *contains* v (see Fig. 8). In other words, if the enlarged box $E(C)$ contains v, then cell C is potentially intersected by N. In this case, C is *fully contained* in N if all 4 vertices of C lie in N; otherwise C is *potentially partially intersected* by N. This is in fact a *range query*: for each cell C, we query with $E(C)$ to find all vertices v inside $E(C)$. To support such queries, we use a preprocessing step to build a KD-tree T on all mesh vertices. Then given an axis-aligned query box Q ($Q = E(C)$), we perform a *range query* on T to find all vertices inside Q. Details on the KD-tree operations are given in Appendix IV (Supplementary Materials).

Now cell sampling should be quite fast, but there is one problem: we have to keep open the local histogram bins for *all* vertices, which requires a large amount of memory. To address this issue, we develop the *plane sweeping* algorithm. The idea is to process the cells C in their sorted order along the sweeping dimension, say the x-dimension, then at any time we only need to keep open for those vertices that are close by and therefore *currently active*. Here is our final algorithm.

Algorithm: Cell Sampling with Sweeping

Step 1: Build a KD-tree on all mesh vertices.

Step 2: Sort the cells C by the smallest x-value of their vertices. This is the queue Q_C of events for cells to

enter the sweeping plane.

Step 3: Sort all mesh vertices by x-values. This is the queue Q_v for vertices v to compute local statistical value (SV) (e.g., entropy) (i.e., to finish) and to release the memory of the histogram.

Step 4: With the order in Q_C (Step 2), process each cell C.

1) Perform a range query on the KD-tree. Put the resulting vertices into two lists of vertices whose neighborhood boxes (a) fully contain C, and (b) potentially partially intersect C.

2) Perform contour spectrum and discrete sampling for the current cell C. Contribute them respectively to the vertices in the two lists (a) and (b) in 1).

If a vertex has no histogram yet, allocate memory for it.

3) Let $x_{\min}$ be the minimum x-value among the vertices of the current cell C. Compute $x_{\min}-t$.

Scan forward on the queue Q_v (Step 3) and find those vertices with x-values $< x_{\min}-t$. Such vertices are no longer active.

Compute local SV for these vertices, and release the memory of their histograms.

Step 5: For the remaining vertices in the queue Q_v (Step 3), compute their local SV and release the histogram memory.

Note that any SV computable from a histogram can be computed. We analyze the time and space complexities in Appendix V (Supplementary Materials). In summary, the overall running time is $O(cN(\log N + M/c + B + S) + NF(B))$, where $F(B)$ is the time to compute the statistic function value from a histogram with B bins; for other symbols see the caption of Table 4. The overall space is optimal $O(N)$.

Remarks:

1. This algorithm is *independent* of the method for sampling from cells. E.g., we can replace the BD-based method by Monte Carlo sampling or other methods. Also, for Monte Carlo sampling, it would be too slow to do it naively in box sampling (Sec. 3.1 before Sec. 3.1.A); for the feasible option of doing it in cell sampling, this algorithm should be used for time and space efficiencies.

2. As mentioned at the beginning of Sec. 3, this algorithm can be slightly modified to work even if the neighborhood box N of size t around each vertex v is replaced by the neighborhood *ball* NB of radius t around v. A tetrahedral cell C is *fully contained* in NB if and only if all 4 vertices of C lie inside NB, which can be easily checked. For partial intersection, the box N contains NB and can be used as a *conservative approximation* — as before, we define C to be potentially partially intersected if it is not fully contained and its axis-aligned bounding box $B(C)$ intersects N, and the idea of Fig. 8 works as before. Thus the KD-tree range query in Step 4 1) works in the same way. Contour spectrum also works for fully contained C. The only difference is that, in Step 4 2), for potentially partially intersected C, each discrete sample p from C is filtered/tested to see if p lies inside the ball NB (instead of the box N), which can be done easily. Note that it is possible for $B(C)$ to intersect N but not NB, but the filtering step still makes the algorithm correct.

3.2 Curvilinear Grids

For curvilinear grids, isoparametric interpolation is used for the cell interior [1]. Note that tetrahedralizing the grid and applying the techniques in Sec. 3.1 is *not* a valid method since this would use linear interpolation and thus violate isoparametric interpolation [10]. So we will work directly on the original grid. As mentioned in Limitations of Sec. 1, we assume that each cell is *convex* and the vertices of each face of a cell are *co-planar* (which are typically true in practice).

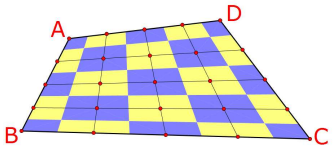


Fig. 9: A curvilinear-grid cell $ABCD$ is cut into convex hexahedra (via the black lines connecting the (red) sample points) by mapping $ABCD$ to a unit cube $\mathscr{C}$, cutting $\mathscr{C}$ by planes through the regular sample points in $\mathscr{C}$ and parallel to the faces of $\mathscr{C}$, and mapping the resulting cubes (and sample points) back to the physical space. For these sample points (shown in red), their corresponding barycentric-dual (BD) cells are colored in yellow or blue.

Computational-Space Sampling with Barycentric-Dual Weighting

We can still use *cell sampling with sweeping* in exactly the same way (the KD-tree method works as well), but since contour spectrum only works for tetrahedral meshes, we need to replace it with (discrete) sampling. Also, the sampling of tetrahedral cells using barycentric axes does not work either. In summary, what we need is to derive a sampling method to regularly sample a hexahedral cell.

For each hexahedral cell, we map its 8 vertices to a unit cube $\mathscr{C}$; the interior of the cell ranges from 0 to 1 in the computational space. Since it is hard to do regular sampling in the physical space, we instead do regular sampling in the computational space, map each sample point back to the physical space, and assign it a suitable volume weight.

First, we cut $\mathscr{C}$ by planes that are parallel to the faces of $\mathscr{C}$ and going through the sample points. The resulting cubes, when mapped back to the physical space, become convex hexahedra (see Fig. 9 for a 2D case). Then for each such convex hexahedron/polytope in the physical space, we perform the barycentric subdivision (BCS) as discussed in Sec. 3.1.A and shown in Fig. 3. The barycentric-dual (BD) cells in Sec. 3.1.A can be applied naturally here and the results are shown in Fig. 9. Note that the BD cells of the sample points can be *trivially* obtained by an *easy local computation*, following the definitions in Sec. 3.1.A. Then for each BD cell we compute its volume and assign it as the volume weight of the related sample. Using a technique similar to that in Sec. 3.1.A, we can also prove the convergence.

Note that the volumes of the BD cells are no longer regular as in tetrahedral meshes (Sec. 3.1.A), and thus we need to compute these volumes individually. However, our main advantage is that the BD cells of the sample points are trivial to obtain *locally*. As a comparison, if we use Voronoi-cell weighting, since the sample points mapped back in the physical space are at arbitrary positions, to get the Voronoi cells we must *explicitly* compute the 3D Voronoi diagram on these points, which is *global* and costly, and we need to do it for each grid cell, making it prohibitively expensive. Therefore, our new method of BD weighting is advantageous in

TABLE 1: Test Datasets.

Dataset	Type	Mesh	Size	# Verts	# Cells
blunt fin	Scalar	Tetrahedral	5.5 MB	41K	187K
post	Scalar	Tetrahedral	16.2 MB	110K	513K
post-vec	Vector	Tetrahedral	18.1 MB	110K	513K
Tpost	Scalar	Tetrahedral	19.1 MB	131K	615K
delta	Scalar	Tetrahedral	33.4 MB	212K	1006K
delta-vec	Vector	Tetrahedral	37.1 MB	212K	1006K
vorts	Scalar	Tetrahedral	350.0 MB	2097K	10241K
cavity	Scalar	Curvilinear	51.3 MB	1167K	1124K
cavity-vec	Vector	Curvilinear	75.9 MB	1167K	1124K

both tetrahedral and curvilinear cases.

3.3 By-Product: Vector Fields

For vector fields, as in common practice, we look at the vector directions and use histogram bins to partition the unit sphere into angular ranges (e.g., [27]). The interpolation is applied to vectors rather than scalar values. Similar to normal vector interpolation from vertices to fragments done in GPU, we perform component-wise linear interpolation on the vectors. These are all common practice and not new. What is newly available is that our scalar-field algorithm *cell sampling with sweeping* can be easily extended to apply as a by-product. For both tetrahedral and curvilinear datasets, it can work in the same way except that contour spectrum only works for tetrahedral scalar fields. We can easily replace this task with discrete sampling along the barycentric axes of the current tetrahedron as discussed before.

Proof of Convergence for Vector Fields

The convergence proof for our sampling in tetrahedral scalar fields at the end of Sec. 3.1.A can be extended to vector fields. Since the vectors of points inside a tetrahedral cell are constructed by linear interpolation from its four vertices, for a vector histogram bin i, the points belonging to bin i form a continuous region; the volume of such region is defined as $\pi_f(i)$. Note that $H_N(i)$, $U_N(i)$ and $L_N(i)$ are defined as before: $H_N(i)$ is the accumulation of the sample points that belong to bin i, weighted by the volume of their BD cells; $L_N(i)$ is the volume of fully contained BD cells; $U_N(i)$ is the volume of partially plus fully contained BD cells. Equations (3) and (4) are applicable as well, showing that the histogram of a vector field also converges to the Hausdorff measure for each histogram bin. Proof for curvilinear grids is similar.

4 RESULTS

We have implemented our approaches in C/C++ and run our experiments on a PC with one 3.4GHz Intel Quad Core i7-2600 CPU, 16GB RAM, nVidia GeForce GTX 570 graphics (1280MB graphics memory), and Linux Fedora 16 OS. The rendering images were produced using the VisIt [13] package. The test datasets are shown in Table 1; they are real-world datasets from scientific applications and have been widely used in the visualization research community.

4.1 Sampling Accuracy and Speed

4.1.1 Comparisons in Computing the Global Histogram

To measure the accuracy of our sampling approach, recall that for tetrahedral scalar fields, our contour spectrum method can compute the *accurate* contribution of a cell C to the histogram bins if C lies *entirely* inside the neighborhood box. To this end, we let the box contain *all* cells *entirely*, i.e., we computed the *global histogram*. On one hand, we applied contour spectrum on each cell to obtain the *accurate* result as the *ground truth*. On the other hand, we applied our sampling approach on each cell, and compared the result with the ground truth; this would give the maximum possible errors for sampling since all cells are counted. We computed the *normalized root mean square error* (*NRMSE*) [2] for the results, shown in Table 2. We compared the run-time and accuracy of our method (weighting by barycentric-dual (BD) cells), against the method of weighting by Voronoi cells[3]. We also compared with *Monte Carlo* sampling on each cell. We see that Monte Carlo sampling converged very slowly and typically had the worst accuracy. On the other hand, our method and Voronoi-cell method produced small errors even with a small k, while our method ran **several thousand times faster** with only slightly worse accuracies. Also, applying contour spectrum on a cell C is faster than generating samples from C. Comparing with Monte Carlo sampling, our method is *better in both speed and accuracy*. In particular, we were about *twice as fast*; this is because our weighting computation is as fast, but our batched sampling and interpolation is done through only offset *additions* while Monte Carlo sampling needs multiplications and divisions. We conclude that our sampling should be the method of choice (additional evaluations comparing against Monte Carlo sampling are given in Sec. 4.1.2 below).

TABLE 2: Sampling run-time (in seconds) and errors in computing the global histogram. Each barycentric axis of a cell is subdivided into k segments by $k+1$ samples. Monte Carlo Sampling generates n_s sample points – the same number as that of the corresponding k. Each result of Monte Carlo Sampling here is an average of 10 runs. Contour Spectrum gives the **ground truth**.

		Dataset: blunt fin		**Dataset: post**	
Method		Run-time	NRMSE	Run-time	NRMSE
Contour Spectrum		0.0316	**0**	0.0834	**0**
Weighting by **BD cells**	k=2	0.0361	0.857%	0.0993	1.348%
	k=3	0.0733	0.505%	0.1985	0.829%
	k=5	0.1984	0.196%	0.5426	0.387%
	k=10	1.0737	0.045%	2.9009	0.105%
Weighting by **Voronoi cells**	k=2	159.62	0.828%	446.92	1.203%
	k=3	364.59	0.492%	1013.7	0.786%
	k=5	1191.2	0.190%	3418.4	0.362%
	k=10	7832.9	0.043%	22627.0	0.098%
Monte Carlo Sampling	n_s=10	0.0819	0.629%	0.2151	1.305%
	n_s=20	0.1720	0.603%	0.4196	1.282%
	n_s=56	0.4232	0.583%	1.1777	1.293%
	n_s=286	2.1184	0.564%	5.7266	1.269%

Note that when $k = 5$ our errors were all less than 0.5%. Since the sampling time is $O(k^3)$ per cell, we fixed k to 5 for all remaining experiments to achieve both efficiency and accuracy.

4.1.2 Comparisons Under Our Overall Algorithm

From Sec. 4.1.1, we see that the Voronoi-cell method is too slow to be practical. The Monte Carlo sampling, on the other hand, seems to still remain competitive to our sampling method — although the running time is twice as ours and it converges very slowly with worse NRMSE's than ours in Table 2, the numerical differences shown there are not very large. Recall that Table 2 is only for computing the global histogram. To further compare their speed

2. It is defined as $\sqrt{(1/n)\sum_{i=1}^{n}(a_i-b_i)^2}$, where a_i, b_i are the values of histogram bin i by sampling and contour spectrum respectively and n the number of bins, normalized by the range of the values of b_i's.

3. We used the Qhull library [4] to compute Voronoi cells.

and accuracy effects on our overall algorithm *cell sampling with sweeping*, we used this algorithm on tetrahedral scalar fields to compute local entropy, with the following variations:
(1) Our sampling with contour spectrum — this is as in Sec. 3.1.C.
(2) Our sampling only — same as **(1)** but use our sampling (rather than contour spectrum) also for fully contained cells.
(3) Monte Carlo sampling with contour spectrum — same as **(1)** but use Monte Carlo sampling in place of our sampling.
(4) Monte Carlo sampling only — same as **(3)** but use Monte Carlo sampling (rather than contour spectrum) also for fully contained cells.
In order to measure the accuracies of **(1)-(4)**, we need the *ground truth*, which we obtained by using the method
(0) Clipping, described below.

Recall from Sec. 3.1.B that our clipping method is *provably accurate*, whose results are the ground truth. Computationally it is related to our *cell sampling* algorithm in Sec. 3.1.C but without using sweeping: for each cell C we query the KD-tree to find the boxes N that C potentially intersects; the contour spectrum on C is contributed to the boxes fully containing C, and for the remaining boxes C is used to perform the clipping. We implemented this approach[4] and call it Clipping.

We used Clipping to compute the ground truth, and compared **(1)-(4)** against the ground truth (the neighborhood box size t was set in the same way as in Sec. 4.2; see there for more details). The run-time and accuracy results are shown in Table 3 top part (ignore the bottom part for now). As seen, our sampling with contour spectrum (Ours_CS) is about **a hundred times faster** than Clipping, and has the best accuracy (NRMSE) among **(1)-(4)**. Also, in general, applying contour spectrum improves both the running time and NRMSE (Ours_CS vs. Ours, and MC_CS vs. MC), as expected, since contour spectrum gives the ground truth and also runs faster than discrete sampling (see Table 2). However, for Tpost the NRMSE stays the same with a very small increase in run-time. This is because there were no fully contained cells for contour spectrum to be applicable (as will be seen in Table 4, the α value (average fraction of the fully contained cells per neighborhood box) is 0 for Tpost). Thus some extra time was spent to compute contour spectrum, but such results were not applicable to improve the overall run-time and accuracy. Comparing between our sampling and Monte Carlo sampling (Ours_CS vs. MC_CS, and Ours vs. MC), our sampling always has a better run-time and NRMSE; in particular, our speed can be *almost twice as fast with a large margin* (e.g., 481.1s vs. 933.5s in Tpost).

More importantly, we want to visually evaluate the visualization qualities generated by the methods **(1)-(4)**, compared against the ground truth produced by Clipping. To this end, for the resulting point-wise local entropy at mesh vertices, we treated the local entropy as a scalar field (call it *local entropy field*), and performed direct volume rendering (see Sec. 4.3 for more details). (Such visualization of the local entropy field has been used [11], [52] and is related to our case study in Sec. 4.4.) The resulting images are shown in Figs. 10 and 11.

In Fig. 10, first look at (a), (c), (e) and the red rectangles: for the left and middle rectangles our image portions (in (c)) are visually the same as the ground truth (in (a)) but Monte Carlo sampling (in (e)) has obviously visible errors in each area; for the right rectangle our image portion (in (c)) is slightly different from the ground truth (in (a)) but is still better than that of Monte Carlo sampling (in (e)). Overall our image quality is obviously better. We observe that the sampling errors are most obvious visually in large cells. It is also very interesting to see that contour spectrum did *not* improve the image quality visually, i.e., visually (b), (c) are the same and (d), (e) are the same. In particular, (c) has a better image quality than (d) even though the corresponding NRMSE's are the opposite (Table 3)! This is because, as said, the sampling errors are most obvious visually in large cells, but large cells are unlikely to be fully contained in a neighborhood box to apply contour spectrum. Therefore, contour spectrum was not able to correct such visually obvious sampling errors in Monte Carlo sampling.

4. We used the Qhull library [4] to compute tetrahedralization.

TABLE 3: Run-time (in seconds) and errors (NRMSE, in %) in computing local entropy (**top part** of the table) and local standard deviation (**bottom part** of the table) using the following methods: **(0)** Clipping, **(1)** our sampling with contour spectrum (Ours_CS), **(2)** our sampling only (Ours), **(3)** Monte Carlo sampling with contour spectrum (MC_CS), and **(4)** Monte Carlo sampling only (MC). Except for Clipping, all methods are variations of our algorithm *cell sampling with sweeping*.

Local Entropy						
Dataset		Clipping	Ours_CS	Ours	MC_CS	MC
blunt	Time	8853.8	81.7	206.6	103.3	306.7
	NRMSE	0	0.083	0.307	0.137	0.415
Tpost	Time	42623.4	481.1	479.7	933.5	932.2
	NRMSE	0	0.067	0.067	0.075	0.075
Local Standard Deviation						
Dataset		Clipping	Ours_CS	Ours	MC_CS	MC
blunt	Time	8851.9	80.8	205.5	102.4	305.7
	NRMSE	0	0.315	0.573	0.521	0.730
Tpost	Time	42612.0	476.2	475.9	928.2	927.9
	NRMSE	0	0.157	0.157	0.182	0.182

In Fig. 11, note that (b), (c) are actually the *same* and so are (d), (e) (recall that $\alpha = 0$ for Tpost). Comparing (a), (c), (e), we see that our sampling result ((c)) is visually (almost) the same as the ground truth ((a)), but we can clearly see the sampling errors/noises in Monte Carlo sampling ((e)). From Figs. 10 and 11 we summarize that Monte Carlo sampling tends to introduce more sampling errors due to its randomness and unstableness; visually such errors are most obvious in large cells, seen as fuzzy noises or sometimes even leading to different (and incorrect) structures/features.

We also repeated the same experiments, but computed the *local standard deviation* instead of local entropy; the results are shown in Table 3 bottom part and Figs. 12 and 13. These results are similar to those of local entropy as discussed above. It is interesting to see that in Fig. 12, in each of (a), (b) and (c) there is a transparent area at the top right region (which looks like a "hole"). This is because the local standard deviation values at the corresponding vertices computed by clipping and our approaches are very close to zero, resulting in near-zero opacity and thus the area looks like transparent. Overall, both the local entropy and the local standard-deviation fields tend to show similar structures. But the local entropy fields usually have better distributions of values, making the features easier to be visually observed. More examples and discussions are given in Sec. 4.3.

In conclusion, for both local entropy and local standard deviation, our sampling can produce obviously better visualization quality than Monte Carlo sampling, regardless of whether contour spectrum is used or not. In addition, our sampling can be almost

TABLE 4: Statistics of some quantities and parameters. N: number of vertices, cN: number of cells, t: neighborhood box size, M: average number of potentially intersecting cells per neighborhood box, α: average fraction of the fully contained cells per neighborhood box, S: number of sample points per cell for cell sampling (with $k+1=6$ samples on each barycentric axis), B: number of histogram bins.

Dataset	N	cN	t	M	α	S	B
blunt fin	41K	187K	0.3	5920	0.252	56	64
post	110K	513K	0.3	9013	0.362	56	64
Tpost	131K	615K	0.6	3880	0	56	64
delta	212K	1006K	0.04	18196	0.239	56	64
vorts	2097K	10241K	1.1	309	0.127	56	64

TABLE 5: Run-time (in seconds) and error comparisons (NRMSE in parentheses) among **S1** Box sampling, **S2** Box sampling with contour spectrum, **S5** Cell sampling with sweeping, and Clipping. Note that Clipping has **0 error**. The top 4 rows and bottom 4 rows are results for computing local entropy and local standard deviation, respectively.

Dataset	Clipping	**S1**	**S2**	**S5**
blunt fin	8853.8	489.1 (0.465%)	690.3 (0.107%)	81.7 (0.083%)
post	39746.1	3346.5 (0.733%)	4409.6 (0.179%)	283.0 (0.126%)
Tpost	42623.4	4533.6 (0.062%)	4550.8 (0.062%)	481.1 (0.067%)
vorts	24019.9	7850.6 (0.034%)	7946.2 (0.007%)	434.4 (0.010%)
blunt fin	8849.2	488.3 (1.681%)	688.1 (0.363%)	80.8 (0.315%)
post	39728.0	3343.0 (2.286%)	4408.4 (0.721%)	282.3 (0.843%)
Tpost	42600.7	4531.8 (0.151%)	4547.9 (0.151%)	476.2 (0.157%)
vorts	24003.9	7847.2 (0.142%)	7942.0 (0.022%)	433.0 (0.013%)

twice as fast with a large margin. Therefore our sampling should be the method of choice.

4.2 Computing Local Histograms/Statistic Functions

Now we compare the performance of our algorithm *cell sampling with sweeping* against other alternative methods in computing local histograms/statistic functions (entropy, standard deviation). We chose the neighborhood box size t as the minimum between the average cell-edge length (based on mesh volume) and the average of the longest edges from all cells. See Appendix III.

4.2.1 Tetrahedral Meshes — Scalar Fields

We show in Table 4 the statistics of some quantities and parameters of the test datasets that are related to the performance of our algorithm. They are useful for our experimental analysis below. We compared the following methods for scalar-field tetrahedral meshes:

S0 Clipping,
S1 Box sampling,
S2 Box sampling with contour spectrum,
S3 Box-based cell sampling (with contour spectrum),
S4 Cell sampling,
S5 Cell sampling with sweeping.

We already described Clipping (**S0**) in Sec. 4.1.2; recall that its results are the ground truth. **S1** box sampling is as described in the beginning of Sec. 3.1, where for each neighborhood box we used $k'=7$ to generate $k'\times k'\times k'=343$ sample points; k' was chosen to generate about the same number of total sample points as our cell sampling methods. For cell location queries, the existing method (e.g. [18]) still needs tree traversals; we used the following approach to achieve decent accelerations for *batched cell locations* by *removing any tree traversal*: we used an octree, which was built by splitting the volume domain until each octree leaf box had side length $\leq t$, or the maximum octree level 7 was reached (or each leaf contained at most one cell), where a cell C is contained in a leaf if the leaf box intersects the axis-aligned bounding box $B(C)$. Also, all leaves are at the same level, so they form a *uniform grid*. Given a sample point p, using its x-, y- and z-values we can compute in $O(1)$ time the (i,j,k) index in this uniform grid, and thus the leaf L of the octree containing p. We can then check the cells in L to see which one contains p. **S2** is the same, but uses contour spectrum for fully contained cells while removing the corresponding sample points. In **S3**, we first grow for each vertex v a list $L1$ of fully contained cells and a list $L2$ of potentially partially intersected cells, by querying the KD-tree with the enlarged box $E(C)$ for all cells C. We then go over the vertices v one by one, use contour spectrum for cells in $L1$ and our sampling in Sec. 3.1.A for cells in $L2$. Note that many cells can be computed more than once because they may intersect with multiple boxes. Methods **S4, S5** are as in Sec. 3.1.C, but **S4** does not use sweeping.

Accuracy and Speed

We used Clipping to compute the ground truth, and compared **S1**, **S2** and **S5** against the ground truth. (Note that **S3, S4, S5** produce the *same* results and have the same accuracies.)

The run-time and accuracy results for local entropy are shown in Table 5 upper half. We see that **S5** is **about a hundred times faster** than Clipping with very small errors. Also, since contour spectrum is faster than sampling (Table 2), we see that computing and tetrahedralizing the intersection regions in Clipping is extremely slow. (Note that **S1** and **S2** had the same accuracy on *Tpost* since its α value is 0 (Table 4), i.e., there were no fully contained cells to apply contour spectrum.) Moreover, our method **S5** had similar yet slightly better accuracy than box sampling (**S1, S2**), with *much faster* running time.

The results for local standard deviation (SD) are shown in Table 5 lower half, which are very similar to those for local entropy. In particular, for each method, the run-times between SD and entropy were almost identical. This is because all computing steps are the same except for computing the target value from the histogram, which is negligible in run-time compared to other steps. Also, their memory footprints are exactly the same. These properties hold for all methods being compared in this section (Sec. 4.2).[5] Thus in the remaining runtime/space analysis we only report the results of local entropy.

Running Time and Memory Space

Next we compare the time and space efficiencies of **S1**–**S5** (Clipping is too slow to compare). The results are shown in Fig. 14. **S1** and **S2** were the slowest due to expensive cell locations. Performing contour spectrum in **S2** was even (slightly) slower (but with more accuracy, as seen (Table 5)). The additional memory for the octree was essential. Box-based cell sampling (**S3**) was usually faster than box sampling methods (**S1, S2**). Cell sampling methods (**S4, S5**) were significantly faster than all the others. However, **S4** used a large amount of memory for the histogram bins of all vertices. With the sweeping approach (**S5**) the memory was greatly reduced. In conclusion, **S5** (our cell sampling with sweeping) is the best in both run-time and memory usage.

5. But note that SD can only be computed from scalar fields due to the difference-square terms in the definition [32].

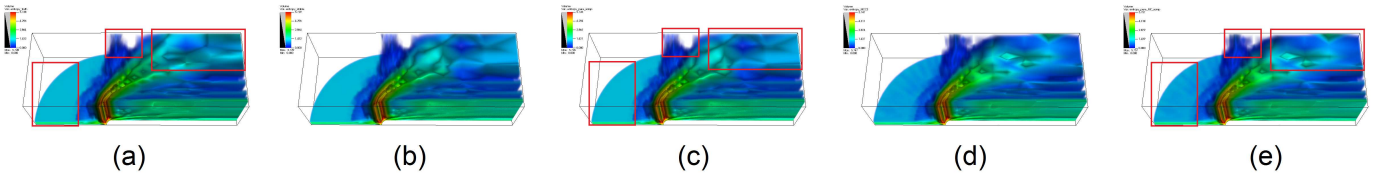


Fig. 10: Direct volume rendering on the local entropy as a scalar field, where the local entropy was computed by (a) Clipping, (b) our sampling with contour spectrum (Ours_CS), (c) our sampling only (Ours), (d) Monte Carlo sampling with contour spectrum (MC_CS), and (e) Monte Carlo sampling only (MC). Note that (a) is the ground truth. The red rectangles indicate the main regions to compare. (Visually (b), (c) are the same and (d), (e) are the same so we do not put red rectangles in (b) and (d).) The dataset is blunt.

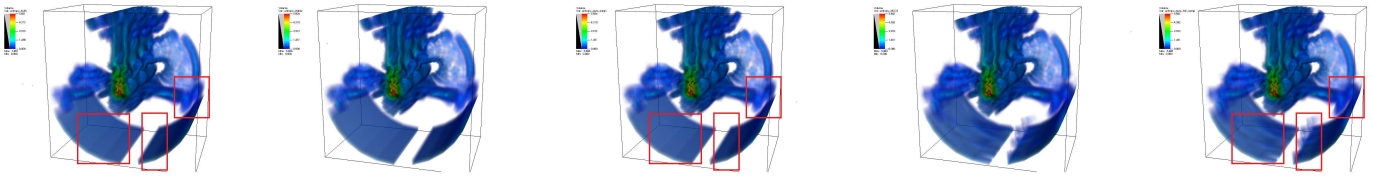


Fig. 11: The caption here is the same as that in Fig. 10 except that the dataset is Tpost.

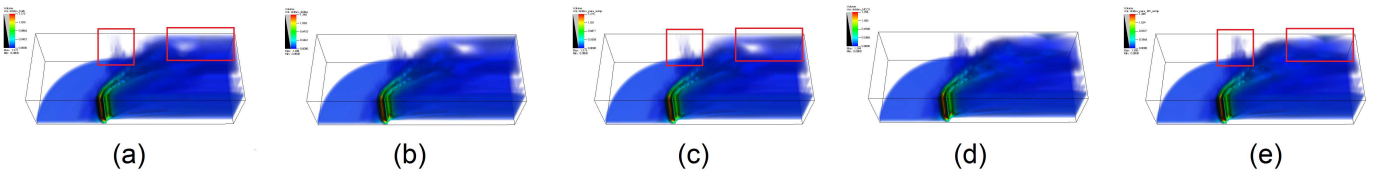


Fig. 12: The caption here is the same as that in Fig. 10 except that the direct volume rendering is on the local standard deviation as a scalar field.

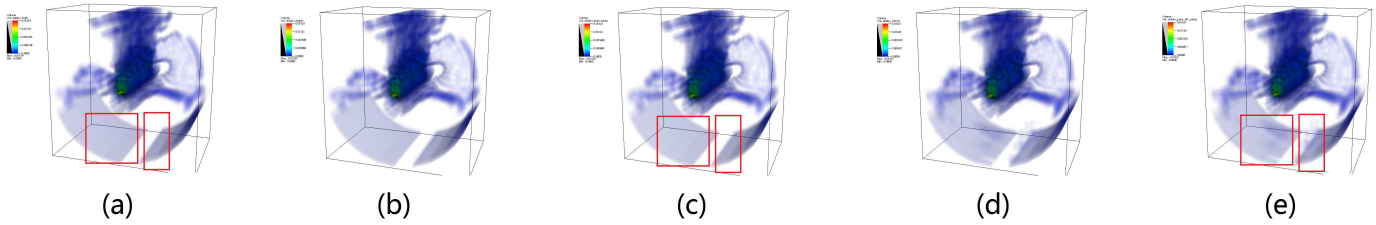


Fig. 13: The caption here is the same as that in Fig. 10 except that the direct volume rendering is on the local standard deviation as a scalar field, and that the dataset is Tpost.

The running times of vorts were faster than some of the smaller datasets (such as delta). This is because the vorts dataset has a very low value of M (the average number of potentially intersecting cells per box) — 309 vs. 18196 for delta; see Table 4. The vertices in vorts are more uniformly spaced. When using our method to choose the box size t, we obtained roughly the spacing of the vertices. Thus each box intersected only a few cells. On the contrary, delta has some big cells occupying most of the volume, making t to cause much more potentially intersected cells per box.

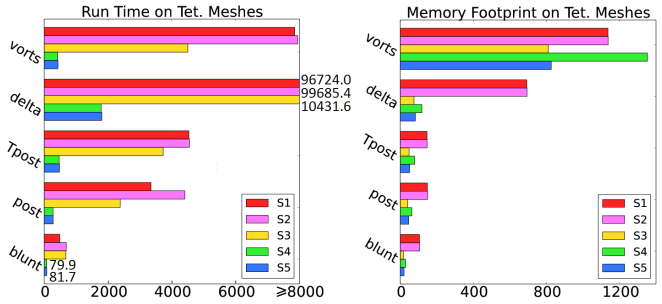
S1 and **S2** on delta were extremely slow (**about 27 hours** each). This was because each box had many potentially intersecting cells (value M), which led to many expensive cell-location operations. To compare, our method **S5** only took **30.1 minutes**.

4.2.2 Tetrahedral Meshes — Vector Fields

For vector-field tetrahedral meshes, note that contour spectrum cannot apply, and we compared the following methods:
V1 Box sampling,
V2 Box-based cell sampling,
V3 Cell sampling,
V4 Cell sampling with sweeping.
The results are shown in Fig. 15. Similar to the scalar-field case, we see that **V4** is efficient in both run-time and memory usage.

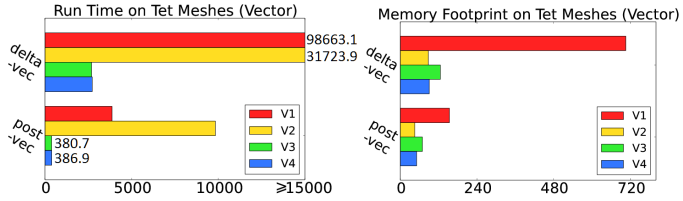
4.2.3 Curvilinear Grids

For curvilinear grids, again contour spectrum is not applicable. Note that box sampling is more difficult than in tetrahedral meshes due to the cell location task. Given a point p, locating the position and computing the parametric coordinate of p in a hexahedral cell is not as efficient as in a tetrahedral cell. This is because the interpolation functions are non-linear, and thus numerical methods such as Newton's method are generally used to solve the equations (e.g. [1]). Since cell location is already slow in tetrahedral meshes and will be even slower and more difficult to implement, we did not implement box sampling here. We compared the following three methods:



(a) (b)

Fig. 14: Performance of various methods for scalar-field tetrahedral meshes. **(a)** Run time in seconds. **(b)** Memory footprint in MB.



(a) (b)

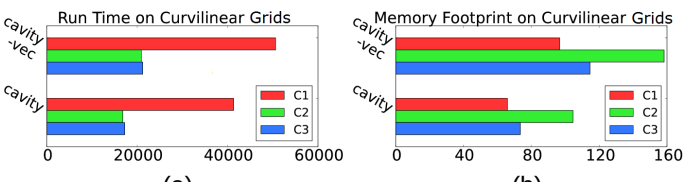
Fig. 15: Performance of various methods for vector-field tetrahedral meshes. **(a)** Run time in seconds. **(b)** Memory footprint in MB.

C1 Box-based cell sampling,
C2 Cell sampling,
C3 Cell sampling with sweeping.

We compared both the scalar-field and vector-field versions; the results are shown in Fig. 16. Similar to the case of tetrahedral meshes, cell sampling (**C2**, **C3**) is much faster than box-based cell sampling (**C1**), while sweeping (**C3**) can greatly reduce the memory footprint.

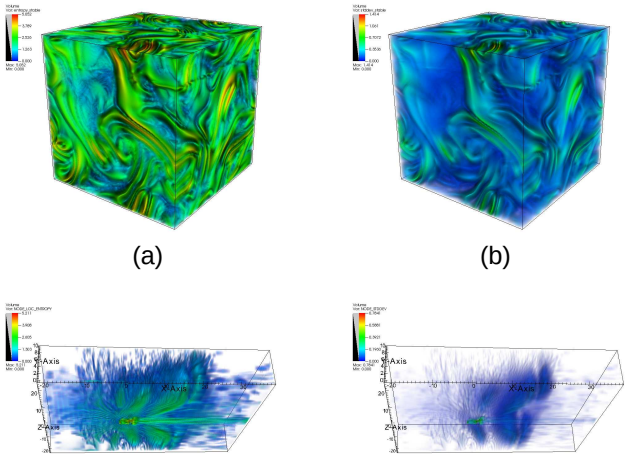
4.3 Visualizing Local Entropy/Standard Deviation

To visualize the point-wise local entropy at mesh vertices, we treated the local entropy as a scalar field (call it *local entropy field*), and performed direct volume rendering. We did the same for local standard deviation (SD). The color and opacity were mapped linearly: low field values were bluish and of low opacity; high field values were reddish and of high opacity — regions of high local entropy (resp. SD) contain more information (resp. variation) and are considered more salient, and hence were given high opacity.



(a) (b)

Fig. 16: Performance of various methods for curvilinear grids. **(a)** Run time in seconds. **(b)** Memory footprint in MB. Both the scalar-field and vector-field versions were compared.



(a) (b) (c) (d)

Fig. 17: Direct volume rendering on the local entropy/SD as a scalar field: **(a)** vorts entropy, **(b)** vorts SD, **(c)** cavity entropy, **(d)** cavity SD.

The results are shown in Fig. 17 (results of 4 additional datasets are given in Appendix VI in the Supplementary Materials (Fig. 4 there)). They basically highlight the salient features in the datasets, which could be helpful in analyzing the data. They tend to capture similar features but may have different distributions. Thus careful tuning of transfer function can lead to better rendering results. In some cases entropy and SD highlight different features. Study in finance and statistics [5] presents potential reasons for preferring entropy to SD.

4.4 Utility Examples: Tetrahedral Vector Fields

In the paper [52], two main methods are given for (regular-grid) vector field visualization: **(I)** the point-wise local entropy (Sec. 3.3) is treated as a scalar field (the *local entropy field* as above) and directly volume rendered; **(II)** the local entropy field and conditional entropy are used for streamline generation. Both methods rely on local entropy computation, which is now enabled by us for tetrahedral meshes. We did a case study on dataset *post-vec*. It was generated by simulating liquid oxygen flowing across a flat plate with a cylindrical post rising perpendicular to the plate. We used the velocity field.

In the paper [52], **(I)** is evaluated against the field of the Frobenius norm of the Jacobian matrix (the Jacobian represents the local vector gradient and is often used to characterize the flow topology). We did the same and show the results in Fig. 18(a)(b). Clearly, our entropy field highlights not only the critical points but also the flow features much better.

There are two phases in **(II)**: (a) putting initial streamlines (using the local entropy field), and (b) incrementally adding more streamlines (using conditional entropy). In the work [52], **(II)** is compared against two standard methods: evenly-spaced seeding [23] and farthest-point seeding [35]; both focus on phase (b) only. They do not consider flow features, and how well they can capture features depends on the initial seeding in phase (a). A common approach is to put initial seeds evenly in space, but this does not consider the field data and is not expected to do well. A more reasonable approach to be compared against, given

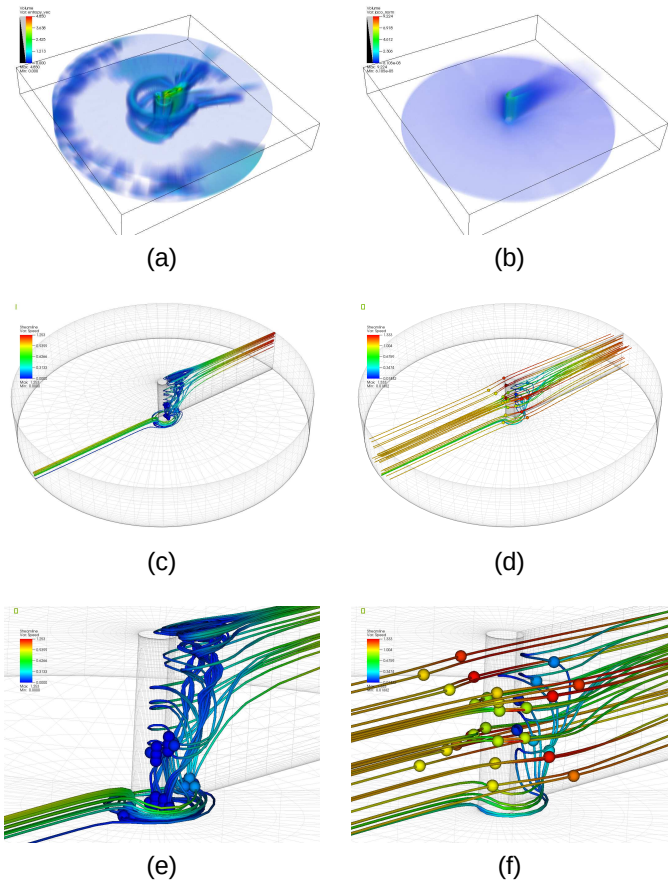


Fig. 18: Case study on *post-vec*: direct volume rendering on **(a)** the local entropy field and **(b)** the Jacobian-norm field, and the resulting streamlines of our initial seeding (**(c)** global view, **(e)** zoom-in view) and of ball seeding (**(d)** global view, **(f)** zoom-in view), integrated in both directions and color-mapped by speed.

that the Jacobian-norm field in Fig. 18(b) already identifies the major activities to be around the center of the dataset, is to put a ball around the dataset center and place initial seeds evenly inside the ball. We call this *ball seeding*. Also, both standard methods ([23], [35]) are for 2D but the 3D evenly-spaced seeding [33] can be used. For us to apply **(II)**, we can also use 3D evenly-spaced seeding [33] in phase (b) instead (since using conditional entropy in **(II)** is complex and not our focus). In summary, the two methods under comparison have the same phase (b), which is *not* related to feature capturing. Thus for feature capturing we only compare phase (a): our initial seeding vs. ball seeding.

Similar to **(II)** in the paper [52], in our initial seeding, we detect local maxima in the entropy field, and discard those vertices whose entropy values are smaller than 0.9 times the maximum entropy value among the candidates. For each remaining vertex v, we put a cube centered at v, and add 7 seeds, one at the cube center and the others at the center of the 6 faces. The results are shown in Fig. 18(c)-(f), where we also compare the results of ball seeding (with the same total number of seeds). As seen, in ball seeding most streamlines go horizontally from left to right and are not very interesting. In our initial seeding, almost all streamlines come from left in the bottom, swirling and rotating up many rounds near the center, and then go right on the top. Clearly they capture much more salient flow features and structures.

5 Conclusions

We have presented novel approaches for computing local histograms/statistic functions in tetrahedral meshes and curvilinear grids. These include mathematical development on sampling and weighting using barycentric duals and proof of convergence, and contour spectrum and clipping for tetrahedral scalar fields. We have also devised a novel algorithm, *cell sampling with sweeping*, to achieve both time and memory efficiencies. As a by-product, this scalar-field algorithm can also work for vector fields. These methods can benefit information theoretic and other distribution-driven analysis. We have also presented a utility case study on tetrahedral vector field visualization. In an on-going work, we are investigating the applications of local entropy for transfer function design in volume rendering for tetrahedral and curvilinear scalar fields, which were not possible before. We will also try to remove the current limitations on curvilinear grids. It is a challenging open question to devise an accurate method (like contour spectrum or clipping) to obtain the ground truth for curvilinear grids.

Acknowledgments

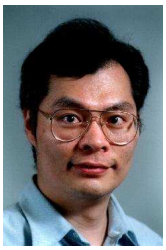
This work was supported in part by DOE Grant DE-SC0004874, program manager Lucy Nowell.

References

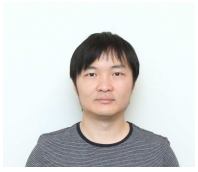
[1] The Visualization Toolkit (VTK) on-line document. http://www.vtk.org/doc/nightly/html/classvtkHexahedron.html#details, extracted Aug. 2017.
[2] S. Bachthaler and D. Weiskopf. Continouous scatterplots. *IEEE Transactions on Visualization and Computer Graphics (Vis '08)*, 14(6):1428–1435, 2008.
[3] C. Bajaj, V. Pascucci, and D. Schikore. The contour spectrum. In *Proc. IEEE Visualization Conference (Vis '97)*, pp. 167–173, 1997.
[4] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, Dec. 1996.
[5] S. R. Bentes and R. Menezes. Entropy: A new measure of stock market volatility? *Journal of Physics: Conference Series*, 394(1):012033, 2012.
[6] A. Bossavit. *Computational electromagnetism variational formulations, complementarity, edge elements*. Academic Press, San Diego, 1998.
[7] S. Bruckner and T. Moller. Isosurface similarity maps. *Computer Graphics Forum (EuroVis 10)*, 29(3):773–782, 2010.
[8] A. Buffa and S. Christiansen. A dual finite element complex on the barycentric refinement. *Comptes Rendus Mathematique*, 340(6):461–464, 2005.
[9] H. Carr, B. Duffy, and B. Denby. On histograms and isosurface statistics. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1259–1266, 2006.
[10] H. Carr, T. Moller, and J. Snoeyink. Artifacts caused by simplicial subdivision. *IEEE Transactions on Visualization and Computer Graphics*, 12(2):231–242, Mar. 2006.
[11] A. Chaudhuri, T.-Y. Lee, B. Zhou, C. Wang, T. Xu, H.-W. Shen, T. Peterka, and Y.-J. Chiang. Scalable computation of distributions from large scale data sets. In *Proc. Symp. Large Scale Data Analysis and Visualization (LDAV)*, 2012.
[12] M. Chen, M. Feixas, I. Viola, A. Bardera, H.-W. Shen, and M. Sbert. *Information Theory Tools for Visualization*. AK Peters/CRC Press, 2016.
[13] H. Childs, E. Brugger, B. Whitlock, J. Meredith, S. Ahern, D. Pugmire, K. Biagas, M. Miller, C. Harrison, G. H. Weber, H. Krishnan, T. Fogal, A. Sanderson, C. Garth, E. W. Bethel, D. Camp, O. Rübel, M. Durant, J. M. Favre, and P. Navrátil. VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In *High Performance Visualization–Enabling Extreme-Scale Scientific Insight*, pp. 357–372. Oct 2012.
[14] L. Codecasa, V. Minerva, and M. Politi. Use of barycentric dual grids for the solution of frequency domain problems by fit. *IEEE Transactions on Magnetics*, 40(2):1414–1419, 2004.
[15] B. Duffy, H. Carr, and T. Moller. Integrating isosurface statistics and histograms. *IEEE Transactions on Visualization and Computer Graphics*, 19(2):263–277, 2013.
[16] H. Federer. *Geometric Measure Thoery*. Springer-Verlag, 1965.
[17] O. Fluck, S. Aharon, D. Cremers, and M. Rousson. GPU histogram computation. In *ACM SIGGRAPH 2006 Research posters*, 2006.

[18] C. Garth and K. Joy. Fast, memory-efficient cell location in unstructured grids for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1541–1550, 2010.
[19] Y. Gu and C. Wang. Transgraph: Hierarchical exploration of transition relationships in time-varying volumetric data. *IEEE Transactions on Visualization and Computer Graphics (Vis'11)*, 17(12):2015–2024, 2011.
[20] S. Gull and J. Skilling. Maximum entropy method in image processing. *Communications, Radar and Signal Processing, IEE Proceedings F*, 131(6):646–659, october 1984.
[21] A. Hatcher. *Algebraic Topology*. Cambridge University Press, 2001.
[22] C. Ip, A. Varshney, and J. JaJa. Hierarchical exploration of volumes using multilevel segmentation of the intensity-gradient histograms. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2355–2363, 2012.
[23] B. Jobard and W. Lefer. Creating evenly-spaced streamlines of arbitrary density. In *Proc. Eurographics Workshop on Visualization in Scientific Computing*, pp. 45–55, 1997.
[24] M. Khoury and R. Wenger. On the fractal dimension of isosurfaces. *IEEE Transactions on Visualization and Computer Graphics (Vis '10)*, 16(6):1198–1205, 2010.
[25] J. Kniss, G. Kindlmann, and C. Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):270–285, 2002.
[26] T.-Y. Lee and H.-W. Shen. Efficient local statistical analysis via integral histograms with discrete wavelet transform. *IEEE Transactions on Visualization and Computer Graphics (Vis '13)*, 19(6):2693–2702, 2013.
[27] P. Leopardi. *Distributing Points on the sphere: Partitions, separation, quadrature and energy*. PhD thesis, The University of New South Wales, 2007.
[28] Y. Lin and G. Medioni. Mutual information computation and maximization using GPU. In *Proceeding of the Computer Vision and Pattern Recognition Workshop*, pp. 1–6, 2008.
[29] C. Lundström, P. Ljung, and A. Ynnerman. Local histograms for design of transfer functions in direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1570–1579, 2006.
[30] C. Lundström, A. Ynnerman, P. Ljung, A. Persson, and H. Knutsson. The α-histogram: Using spatial coherence to enhance histograms and transfer function design. In *Proc. EuroVis*, pp. 227–234, 2006.
[31] R. Maciejewski, I. Woo, W. Chen, and D. Ebert. Structuring feature space: A non-parametric method for volumetric transfer function generation. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1473–1480, 2009.
[32] P. S. Mann. *Introductory Statistics*. Wiley, 2012.
[33] O. Mattausch, T. Theußl, H. Hauser, and E. Gröller. Strategies for interactive exploration of 3D flow using evenly-spaced illuminated streamlines. In *Proc. Spring Conf. Computer Graphics (SCCG '03)*, pp. 213–222, 2003.
[34] T. McLoughlin, R. S. Laramee, R. Peikert, F. H. Post, and M. Chen. Over two decades of integration-based, geometric flow visualization. *Comput. Graph. Forum*, 29(6):1807–1829, 2010.
[35] A. Mebarki, P. Alliez, and O. Devillers. Farthest point seeding for efficient placement of streamlines. In *Proc. IEEE Visualization (Vis '05)*, pp. 479–486, 2005.
[36] J. Munkres. *Elements of Algebraic Topology*. Westview Press, 1996.
[37] C. Nugteren, G.-J. van den Braak, H. Corporaal, and B. Mesman. High performance predictable histogramming on GPUs: exploring and evaluating algorithm trade-offs. In *GPGPU-4: Proc. Workshop General Purpose Processing on Graphics Processing Units*, pp. 1:1–1:8, 2011.
[38] N. Oliver, B. Rosario, and A. Pentland. A Bayesian computer vision system for modeling human interactions. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 22(8):831–843, 2000.
[39] C. Rocchini and P. Cignoni. Generating random points in a tetrahedron. *J. Graphics Tools*, 5(4):9–12, 2000.
[40] S. Roettger, M. Bauer, and M. Stamminger. Spatialized transfer functions. In *Proc. EuroVis*, pp. 271–278, 2005.
[41] C. Scheidegger, J. Schreiner, B. Duffy, H. Carr, and C. Silva. Revisiting histograms and isosurface statistics. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1659–1666, 2008.
[42] T. Scheuermann and J. Hensley. Efficient histogram generation using scattering on GPUs. In *Proc. Symp. Interactive 3D graphics and games (I3D '07)*, pp. 33–37, 2007.
[43] L. Selle, G. Lartigue, T. Poinsot, R. Koch, K.-U. Schildmacher, W. Krebs, B. Prade, P. Kaufmann, and D. Veynante. Compressible large eddy simulation of turbulent combustion in complex geometry on unstructured meshes. *Combustion and Flame*, 137(4):489–505, 2004.
[44] M. A. Selver and C. Guzelis. Semiautomatic transfer function initialization for abdominal visualization using self-generating hierarchical radial basis function networks. *IEEE Trans. Visualization and Computer Graphics*, 15(3):395–409, 2009.
[45] R. Shams, P. Sadeghi, R. Kennedy, and R. Hartley. Parallel computation of mutual information on the GPU with application to real-time registration of 3D medical images. *Computer Methods and Programs in Biomedicine*, 99(2):133–146, 2010.
[46] C. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.
[47] C. Silva, J. Comba, S. Callahan, and F. Bernardon. A survey of GPU-based volume rendering of unstructured grids. *Brazil. J. Theo. Appl. Comput. (RITA)*, 12(2):9–29, 2005.
[48] S. Takahashi, I. Fujishiro, Y. Takeshima, and T. Nishita. A feature-driven approach to locating optimal viewpoints for volume visualization. In *Proc. IEEE Visualization*, pp. 495–502, 2005.
[49] S. Tenginakai, J. Lee, and R. Machiraju. Salient isosurface detection with model-independent statistical signatures. In *Proc. IEEE Visualization*, pp. 231–238, 2001.
[50] D. Thompson, J. Levine, J. Bennett, P.-T. Bremer, A. Gyulassy, V. Pascucci, and P. Pebay. Analysis of large-scale scalar data using hixels. In *Proc. IEEE Symp. Large Data Analysis and Visualization (LDAV '11)*, pp. 23–30, 2011.
[51] Y. Wang, W. Chen, J. Zhang, T. Dong, G. Shan, and X. Chi. Efficient volume exploration using the Gaussian mixture model. *IEEE Transactions on Visualization and Computer Graphics*, 17(11):1560–1573, 2011.
[52] L. Xu, T.-Y. Lee, and H.-W. Shen. An information-theoretic framework for flow visualization. *IEEE Trans. Visualization and Computer Graphics (Vis '10)*, 16(6):1216–1224, 2010.

Bo Zhou is currently a PhD. student in the Department of Computer Science and Engineering at New York University, where he also received his MS. in Computer Science. His research interests include large-scale data visualization and analysis, probabilistic and information-theoretic machine learning, computational geometry, and high-performance algorithms and data structures.



Yi-Jen Chiang is an Associate Professor in the Department of Computer Science and Engineering at New York University. He received his MS. and PhD. in Computer Science from Brown University. His research interests are in big data computation, analysis and visualization, including out-of-core techniques, information-theoretic data analysis and visualization, computational geometry, computational topology, graphics compression, robot motion planning, and high-performance algorithms and data structures. He received an NSF CAREER award and a Best Paper Award in Eurographics (EG). He has served as an International Program Committee (IPC) member for conferences including IEEE Vis, EuroVis, VDA, and ISVC, and as a paper co-chair for VDA. His research has been supported by NSF and DOE.



Cong Wang got his PhD. in Computer Science from the Department of Computer Science and Engineering at New York University. He is currently working in Google's Geo Platform team. His research interests include data visualization, data analysis, and robotics motion planning.