
Proceedings Paper:
Dilys, D., Carr, H. and Boeing, S. (2025) Lagrangian Simulation Volume-Based Contour Tree Simplification. In: 2025 IEEE Topological Data Analysis and Visualization (TopoInVis). 2025 IEEE Workshop on Topological Data Analysis and Visualization (TopoInVis), 02-03 Nov 2025, Vienna, Austria. IEEE. ISBN: 979-8-3315-7992-0.

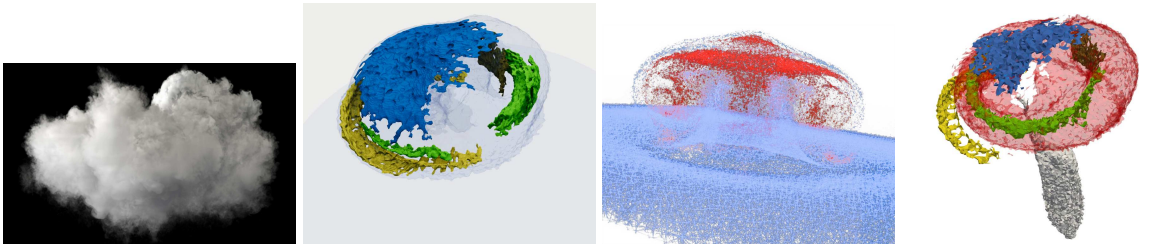
https://doi.org/10.1109/TopoInVis68599.2025.00014

Lagrangian Simulation Volume-Based Contour Tree Simplification

Domantas Dilys*
University of Leeds
School of Computer Science

Hamish Carr†
University of Leeds
School of Computer Science

Steven Boeing‡
University of Leeds
School of Earth and Environment



(a) 1024^3 grid volume rendering (b) 512^3 grid segmentation (c) 2M parcel Delaunay mesh (d) 2M parcel segmentation

Figure 1: Visualisation and topological analysis of Lagrangian simulations often have to rely on costly resampling onto a grid (a), (b). We perform topological data analysis directly from the parcel neighbourhoods (c) faster, and with good quality (d).

Abstract

Many scientific and engineering problems are modelled by simulating scalar fields defined either on space-filling meshes (*Eulerian*) or as particles (*Lagrangian*). For analysis and visualization, topological primitives such as contour trees can be used, but these often need simplification to filter out small-scale features. For parcel-based convective cloud simulations, simplification of the contour tree requires a volumetric measure rather than persistence. Unlike for cubic meshes, volume cannot be approximated by counting regular vertices. Typically, this is addressed by resampling irregular data onto a uniform grid. Unfortunately, the spatial proximity of parcels requires a high sampling frequency, resulting in a massive increase in data size for processing. We therefore extend volume-based contour tree simplification to parcel-in-cell simulations with a graph adaptor in `Viskores` (`VTK-m`), using Delaunay tetrahedralization of the parcel centroids as input. Instead of relying on a volume approximation by counting regular vertices – as was done for cubic meshes – we adapt the 2D area splines reported by Bajaj et al. [1], and Zhou et al. [37]. We implement this in `Viskores` (formerly called `VTK-m`) as prefix-sum style hypersweeps for parallel efficiency and show how it can be generalized to compute any integrable property. Finally, our results reveal that contour trees computed directly on the parcels are orders of magnitude faster than computing them on a resampled grid, while also arguably offering better quality segmentation, avoiding interpolation artifacts.

Index Terms: Visualization, Topological Analysis, Contour Tree, 3D Lagrangian Segmentation, Geometric Measures, Parcel-in-Cell

1 Introduction

Scientific computing and engineering depend on fluid simulations, which are increasingly computed using *Lagrangian* techniques for efficiency instead of *Eulerian* rectilinear grids. Once the data are computed, it must still be visualized, understood, and analyzed.

*e-mail: D.Dilys@leeds.ac.uk
†e-mail: H.Carr@leeds.ac.uk
‡e-mail: S.Boeing@leeds.ac.uk

Contour trees help to visualize scientific datasets and can be computed directly on irregular tetrahedral meshes. However, implementations have focused on regular rectilinear grids. For topological simplification, the standard practice to compute the volume has been by simple vertex count approximation, which is a cheap operation and converges rapidly as the grid resolution increases [10].

Parcel-in-Cell (PIC) are hybrid Lagrangian-Eulerian simulations, tracking fluid *parcels* of nonzero volume (unlike *particles*), driven by an underlying fixed-resolution coarse *simulation grid*. Visualization of PIC data needs an additional post-processing step, computing the weighted contributions of every parcel at each point on the *interpolation grid* (much finer than the *simulation grid*). This increases costs in storage, computation, visualization, and analysis.

Because parcels are often tightly packed in space, the *Nyquist limit* [33] requires a high sampling rate, and working directly with the parcels is preferable. Since PIC simulations exchange material between adjacent parcels, we construct the Delaunay tetrahedralization of the parcel centroids and use this as an irregular mesh.

The approximation of volume by vertex count is only appropriate for regular meshes [6]. For irregular meshes, we revert to the observation by Pascucci et al. [1] that geometric properties, such as volume, can be expressed by sets of polynomial coefficients.

Initial `VTK-m` (now `Viskores`) [27] contour tree implementation made the strong assumption that vertex degree was bounded (as is normal in regular grids) and applied optimizations on this basis. We report on the changes necessary to compute contour trees for any irregular mesh, complete with hypersweep-based summation of polynomial coefficients, branch decomposition, and visualizations.

We show that working directly from the parcels is more efficient than resampling at high resolutions, and also more accurate.

We start with a brief description of the problem in meteorology Sec. 2, and a summary of the differences between grid-based and Parcel-in-Cell simulations in Sec. 3, followed by a summary of the key developments in computational topology relevant to this paper in Sec. 4. We will then be able to state the problem being solved in this paper in Sec. 5. We will then describe the details of the polynomial coefficients necessary to compute volume in a tetrahedral mesh in Sec. 6, and how to adapt the parallel hypersweep to them in Sec. 7. Once this is complete, we will discuss the implementation details for this pipeline in the `Viskores` toolkit in Sec. 8, and evaluate the performance and quality in Sec. 9. Finally, we will summarize our results and speculate on future work in Sec. 10.

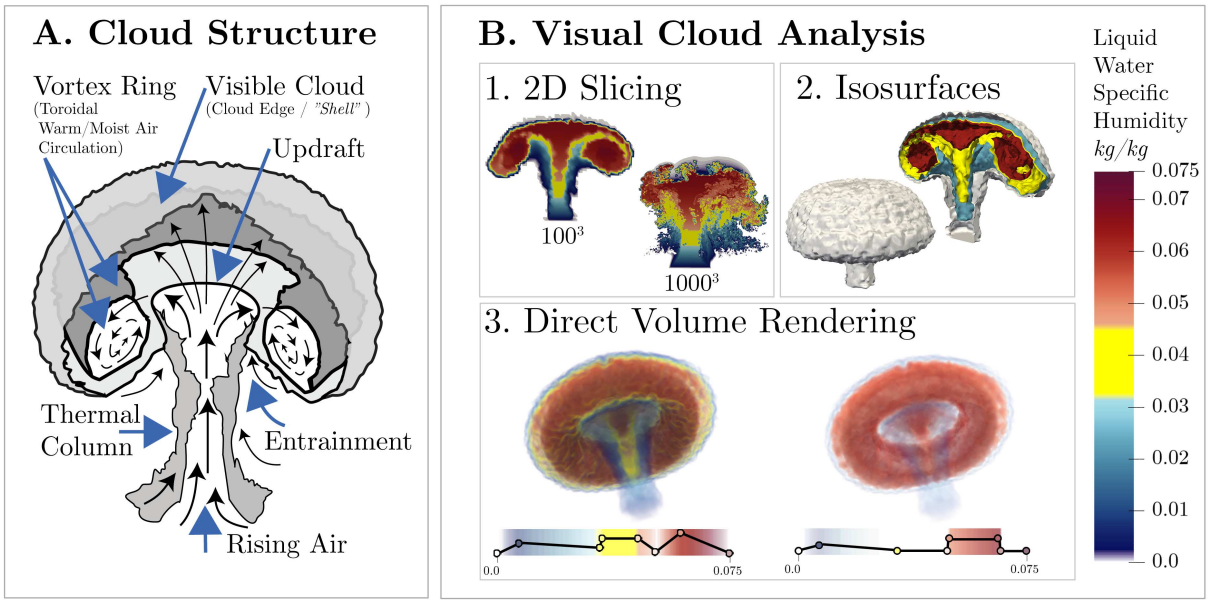


Figure 2: Current visual cloud structure analysis techniques. *2D slicing* (B.1) is a common way for inspecting vortex rings (A), but features tend to spread out at higher resolutions (e.g. at 1000^3 grids). *Isosurfaces* (B.2) often hide structures of interest with their outer shell (left), while cross-sections (right) show the characteristic Φ profile but miss the full 3D form. *Direct Volume Rendering* (B.3) is somewhat better, but the blurry nature of rendering makes it hard to perceive features of interest (left), and transfer function design is complex (bottom-right).

2 Meteorological Background

Cloud formation depends on transport of air from the surface to upper levels. This *thermal column* of warm air can curl up into *vortex rings* as it punches into colder surrounding air (Fig. 2 A).

Entrainment, a process of dry surrounding air entering a moist cloud, influences cloud droplet distribution [3] and cloud lifespan [25]. This mixing process can speed up when vortex rings are present [14]. If the thermal cools enough during the intake of drier air, the cloud can dissipate. Therefore, the cloud life-cycle depends on the entrainment rate, which itself is an active area of research [4].

Vortex rings and related mixing processes are studied to understand entrainment better. High-resolution data are essential for capturing local variation and detailed flow structure [19]. Vortex rings are often identified *qualitatively*, through visualization (Fig. 2 B). In 2D cross-sections, they appear as Φ-shaped structures, with the thermal column in the middle and circles on each side. In 3D, they form tori that trap air with distinct scalar values often unknown beforehand. Both views are unreliable, as the ring's appearance varies with transfer functions, simulation parameters, and resolution [5].

Although these visualizations are effective at small scale for deriving insight, the long-term goal is reliable *quantitative* detection over large regions, such as west Africa. Moreover, spatial metrics, such as the total volume of the vortex ring, are useful for modeling. One of the possible solutions for a more robust analytic pipeline is to apply contour tree analysis (Sec. 4). Before we approach this, however, we must look at one of the key characteristics of modern meteorological simulations: the use of parcel-in-cell simulations.

3 Parcel-in-Cell Simulations

Historically, meteorological simulation started with Eulerian approaches on regular grids in 2D and 3D, as with many scientific and engineering problems [15]. Regular grids have uniform resolution, even in inactive regions. While effective, ever-finer scales require increasing resolution, driving the computational cost up for *Large-Eddy Simulation* (LES) and *direct numerical simulation*.

Regular grids are not the only way. Lagrangian methods track individual flow elements, focusing on active regions. Early examples include *contour advection* – following the fluid boundaries (like the outline of an ink drop). Later, to combine the strengths of both, hybrid methods were derived, such as the *Parcel-in-Cell* (PIC), where volumetric *parcels* are tracked through an underlying *simulation grid*. The parcels carry prognostic information on dynamic (vorticity) and thermodynamic (heat, moisture) properties of the flow at a scale that is finer than the simulation grid. The *Elliptical Parcel-in-Cell* (EPIC) variant improves spatial packing (incompressibility) and the accuracy of advection by using deformable ellipses instead of spheres [19]. Earlier studies [19, 20] showed that EPIC compares favorably with gridded LES models run at even slightly higher resolutions, with EPIC producing much less diffuse vortex rings than LES [20] because diffusion can be explicitly controlled in EPIC.

Advances in PIC and EPIC have not always been accompanied by advances in the visualization tools for examining and analyzing the data. As a result, visualization is currently performed largely by resampling the simulation to a high-resolution grid, then invoking existing tools: as we will see in Sec. 9, this is not the ideal solution.

4 Contour Tree Analysis

Although both isosurface extraction [24] and direct volume rendering [23, 16] assist in analysing cloud formation, selecting suitable isosurfaces is difficult, as the features of interest are inside the cloud and are likely to be occluded by exterior structures (Fig. 2 B.2).

One solution is the *flexible isosurface* [11], which uses the *contour tree* to select connected components of isosurfaces (*contours*), allowing finer-grained choice of what is seen. Moreover, geometric properties of contours such as volume, surface area, or integrals of secondary properties can be pre-computed with the tree.

A Reeb graph [31] is the quotient space under continuous contraction of connected components of inverse images of an isovalue h in a function f on some manifold M. Most often, the manifold M is a mesh from a simulation, and is a compact subset of either $\mathscr{R}^2$ or $\mathscr{R}^3$. In this case there are no cycles, and the Reeb graph is referred to as a *contour tree*, composed of *supernodes* and *superarcs*, with the mesh vertices as *regular nodes* along the superarcs.

The contour tree is useful for extracting isosurfaces [1] or individual contours, but also for finding features and their relationships based on the extrema and critical points [11]. They can be computed efficiently in serial for simplicial meshes [11], non-simplicial meshes [29, 9] or by choosing a suitable *topology graph* that captures neighborhood relationships [9, 13]. More recently parallel algorithms have been described for shared memory [21], PRAM [12], distributed [29] or hybrid distributed/PRAM [7] models.

As data sizes increase, so do contour trees, which can be simplified by discarding superarcs with small height [30], or by using geometric properties such as volume in a similar fashion [11].

While edge height is easy to calculate, other geometric properties are not. Bajaj et al. [1] showed that geometric properties such as volume, surface, etc. are functions of the isovalue, and that in simplicial meshes, are often per-simplex polynomial splines. Then, one can compute the function for every possible isovalue in advance. However, their development of the 3D are splines was slightly imprecise, and did not apply Federer's co-area formula [18] to adjust for gradient which was identified as necessary [32], with Zhou et al.[37] confirming this and correcting the missing details.

Later work [11] established that these computations could be extended from functions for the entire domain M to functions with respect to individual points in the contour tree, and could be computed efficiently by sweeping through the tree, one regular node (mesh vertex) at a time. Computation of the coefficients involved taking cells incident to each node, subtracting out their coefficients for the value range before the vertex was swept past and adding the coefficients for the value range after the vertex was swept past.

This work also showed that for a regular mesh, the regular node count on each superarc was a cheap and effective approximation of volume. Pascucci [28] showed how to compute the *Betti numbers* [28] for edges on the contour tree, which could potentially detect the characteristic toroidal configuration of vortex rings.

In recent years, efforts have focused on adding contour tree analysis to the principal topology toolkits: `TTK` (*Topology Tool Kit*) [36] (now incorporated into `ParaView`), and `Viskores` [27]. Either can be used for contour tree analysis, with `Viskores` performing better at scale, while `TTK` already supports irregular meshes.

Hristov et al. [22] demonstrated efficient parallel computation in `Viskores` for arbitrary contour tree properties using prefix-sums. Their approach uses a variation on rake-and-contraction [26] called the *hypersweep*, however, was applied only to compute volume by counting regular nodes. In contrast, there is no explicit mechanism in `TTK` for computing properties through the contour tree sweep.

In summary, parallel contour tree algorithms exist, but computation of geometric properties on irregular meshes is incomplete.

5 Problem Statement

In applying contour tree analysis to meteorological simulations of cloud formation, the obvious solution is to resample from the parcel-in-cell simulation to a rectilinear mesh, then run the existing tools, either in `TTK` or in `Viskores`. However, a problem arises - what is the appropriate sampling rate, and how much does it cost?

The parcel centroids are not uniformly distributed (Sec. 3). This means that sampling resolution on a regular grid must be driven by the closest pair of parcels. Following the Nyquist sampling theorem [33], we would need a resolution of no more than half of this distance. For our PIC simulation, which has around $2,000,000$ parcels, this argues for a $4,096^3 = 68,719,476,736$ sampling grid resolution, and $> 256GB$ of RAM just to store a single variable.

Previous experience has shown that the memory footprint of contour tree analysis is $\approx 200B$ or more per sample, implying the need for $> 10TB$ of RAM if the computation is performed on a single machine. At this point, we started asking if it was wiser to work directly with the PIC representation rather than re-sampling it.

Applying the knowledge derived from the previous work outlined above, we can see that the first step is to choose a suitable topology graph based on the parcel-in-cell simulations. Since these simulations already depend on tracking which parcels are near each other, the natural choice is to use the Delaunay triangulation, which we can then represent as an irregular mesh in `Viskores`.

Although all of the other steps are on principle solved, the remaining obstacle is the computation of volume, which is the property of interest to the meteorologists. For an irregular mesh, such as a Delaunay triangulation, counting regular nodes is not an effective approximation, so Sec. 6 will develop the formulae necessary to compute volume correctly in an irregular mesh such as Delaunay.

Once we have done this, we will give the implementation details that were necessary to add this to the `Viskores` library in Sec. 8, then compare the rectilinear and PIC-based results in Sec. 9.

6 Polynomial Coefficients

We saw in Sec. 4 that area and volume are polynomial splines in the isovalue h [1] and that the correct derivation needs Federer's co-area formula [18] for area integration [37]. The polynomial must be expressed in standard form (such as ah^2+bh+c) for summing coefficients (e.g. a,b,c) with a hypersweep [22] (Sec. 4). Instead of expanding B-splines [37] recursively into the standard form, we develop a direct geometric construction, making explicit how each coefficient arises from the tetrahedral mesh itself rather than treating them as abstract results of the spline formulation.

Assume a tetrahedron has vertices A,B,C,D with function values $h_A < h_B < h_C < h_D$ (Fig. 3). Barycentric contours in a tetrahedron are parallel planes. Where contours meet tetrahedron edges, the angle is constant and is fixed by the contours intersecting with the edge dihedral. Vertices E,F are defined by linear interpolation on edges AD,AC by the triangular contour BEF at $h = h_B = f(B)$. We get $Area(BEF)$ from half the cross product of FB,FE. We can then compute ϕ, the angle at F in BEF, or more importantly, $\sin(\phi)$. Similarly, we compute vertices G,H of contour CGH at $h_C = f(C)$, $Area(CGH)$, and $\sin(\theta)$. The scalar triple product is used to compute $Volume(ABEF)$ and $Volume(DCGH)$.

We consider contours in the range $h_A < h < h_B$: in Fig. 3, the contour is a single triangle JKL, where J,K,L are linearly interpolated along AB,AE,AF by the parameter $r = \frac{h-h_A}{h_B-h_A}$.

It then follows that:

$$\begin{aligned} Area(JKL) &= r^2 Area(BEF) \\ &= \frac{Area(BEF)}{(h_B-h_A)^2}(h^2-2h_Ah+h_A^2) \\ Volume(AJKL) &= r^3 Volume(ABEF) \\ &= \frac{Volume(ABEF)}{(h_B-h_A)^3}(h^3-3h_Ah^2+3h_A^2h-h_A^3) \end{aligned}$$

For isovalues in the range $h_C < h < h_D$, the position is symmetric, and we parameterize on $t = \frac{h_D-h}{h_D-h_C}$ to get:

$$\begin{aligned} Area(MNO) &= \frac{Area(CGH)}{(h_D-h_C)^2}(h^2-2h_Dh+h_D^2) \\ Volume(DMNO) &= \frac{Volume(DCGH)}{(h_D-h_C)^3}(h_4^3-3h\cdot h_D^2+3h^2h_4-h^3) \end{aligned}$$

However, since we are actually interested in the volume to the left of $\triangle MNO$, we convert the volume by subtraction:

$$Volume(ABCMNO) = Volume(ABCD) - Volume(DMNO)$$

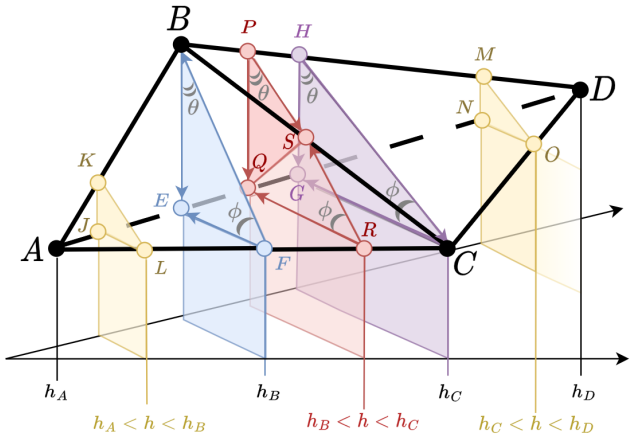


Figure 3: Tetrahedron $ABCD$ with values $h_A < h_B < h_C < h_D$. Barycentric interpolation guarantees that all contours are planar and parallel, with constant angles θ, ϕ. Area at isovalue h is computed with $\triangle JKL$, $\triangle PQS$ and $\triangle QRS$, or $\triangle MNO$, and volume is integrated from area as described in the text.

This leaves the middle section, with isovalues $h_B < h < h_C$. We know $Volume(ABEF)$ and want the $Volume(BEFPQRS)$. We consider the area of quad $PQRS$, noting that angles θ, ϕ are fixed by the edge dihedral, and using $PQ = |PQ|, PS = |PS|$, and so forth. Now, we know that P and Q are linearly interpolated with parameter $s = \frac{h-h_B}{h_C-h_B}$ and $1-s = \frac{h_C-h}{h_C-h_B}$ between B and H respectively, so:

$$\begin{aligned} PQ &= s \cdot HG + (1-s)BE \\ PS &= s \cdot HC \end{aligned}$$

$$\begin{aligned} PQ \cdot PS &= \left(\frac{HG-BE}{h_C-h_B}h + \frac{h_C BE - h_B HG}{h_C-h_B}\right) \\ &\quad \cdot \left(\frac{HC}{h_C-h_B}h + \frac{-h_B HC}{h_C-h_B}\right) \\ &= \frac{(HG-BE)HC}{(h_C-h_B)^2}h^2 \\ &\quad + \frac{2h_B HG \cdot HC + (h_C-h_B)BE \cdot HC}{(h_C-h_B)^2}h \\ &\quad + \frac{\left(h_B^2 HG - h_B h_C BE\right)HC}{(h_C-h_B)^2} \end{aligned}$$

Repeating the process for $\triangle QRS$ gives us a similar set of terms:

$$\begin{aligned} RQ &= s \cdot EF + (1-s)GC \\ RS &= (1-s) \cdot BF \\ RQ \cdot RS &= \left(\frac{EF-GC}{h_C-h_B}h + \frac{h_C GC - h_B EF}{h_C-h_B}\right) \\ &\quad \cdot \left(\frac{BF}{h_C-h_B}h_C - \frac{BF}{h_C-h_B}h\right) \\ &= \frac{(GC-EF)\cdot(-BF)}{(h_C-h_B)^2}h^2 \\ &\quad + \frac{-2h_C BF \cdot EF + (h_C+h_B)GC \cdot BF}{(h_C-h_B)^2}h \\ &\quad + \frac{\left(h_C^2 EF - h_B h_C GC\right)BF}{(h_C-h_B)^2} \end{aligned}$$

Finally, we use these to compute:

$$\begin{aligned} Area(PQRS)(h)) &= Area(\triangle PQS) + Area(\triangle QRS) \\ &= \frac{1}{2}\sin(\theta)PQ \cdot PS + \frac{1}{2}\sin(\phi)RQ \cdot RS \\ &= \alpha h^2 + \beta h + \gamma \end{aligned}$$

$$\begin{aligned} \alpha &= \frac{\sin(\theta)(HG-BE)HC - \sin(\phi)(GC-EF)BF}{2(h_C-h_B)^2} \\ \beta &= \frac{\sin(\theta)(2h_B HG \cdot HC + (h_C-h_B)BE \cdot HC)}{2(h_C-h_B)^2} \\ &\quad + \frac{\sin(\phi)(-2h_C BF \cdot EF + (h_C+h_B)GC \cdot BF)}{2(h_C-h_B)^2} \\ \gamma &= \frac{\sin(\theta)\left(h_B^2 HG - h_B h_C BE\right)HC}{2(h_C-h_B)^2} \\ &\quad + \frac{\sin(\phi)\left(h_C^2 EF - h_B h_C GC\right)BF}{2(h_C-h_B)^2} \end{aligned}$$

For volume, we apply Federer's co-area formula [18] and integrate with respect to h. This computes the interval volume between two isosurfaces by applying a correction factor of the inverse of the gradient magnitude [17]. In a barycentric interpolant, gradient magnitude is constant, and is the difference between isovalues h_B, h_C divided by the perpendicular distance between BEF and CGH, using the normal form of plane BEF, $\vec{n} = \frac{EF \times BF}{|EF \times BF|}$ and points B, H, i.e. $\delta = \vec{n} \cdot B - \vec{n} \cdot H$. Since δ is independent of h, it can be moved outside the integral. The volume polynomial for $BEFPQRS$ is then:

$$Volume(BEFPQRS)(h) = \frac{\alpha\delta}{3}h^3 + \frac{\beta\delta}{2}h^2 + \gamma\delta h + D$$

where D, the constant of integration can be determined by observing that for $h = h_B$, the volume must be 0, since we are interested in the Δvolume (change since the last vertex), and back-substituting.

Once we have the correct coefficients, we can compute local (per-tetrahedron) coefficient changes (deltas) in a parallel preprocessing step. Then, these are summed efficiently in the polynomial hypersweep (Sec. 7) to obtain the volume for each superarc.

7 Polynomial Hypersweep

We saw in Sec. 4 that the existing pipeline was implemented for regular grids, and that volume was approximated by the regular node count. In Sec. 6, we developed the correct polynomial coefficients.

Since `Viskores` is based on the PRAM model, we consider how to extend hypersweeps [22] to update the polynomial coefficients. For each tetrahedral cell K incident to vertex v, the sweep subtracts coefficients $k_1, k_2, \ldots, k_n$ before the vertex is swept, then adds coefficients $K_1, K_2, \ldots, K_n$ afterwards. This is equivalent to adding $\delta_1 = K_1 - k_1, \ldots, \delta_n = K_n - k_n$, which can be computed independently for each vertex in each cell.

The solution is then to compute the deltas for each cell in the mesh at each vertex (wlog, in the downwards direction). Assign the deltas to the vertex, and sum all deltas for each vertex to produce a single set of deltas per vertex. The correct coefficients in each regular arc can then be computed as a prefix sum using a hypersweep.

This is not yet complete, as hypersweeps may reverse the sweep direction. The existing code subtracts node count from total count for the entire data set, then adds one because the origin supernode of each superarc is counted as part of the superarc. For polynomial coefficients, it is necessary to modify the approach so that deltas are stored at each supernode as well as at each superarc.

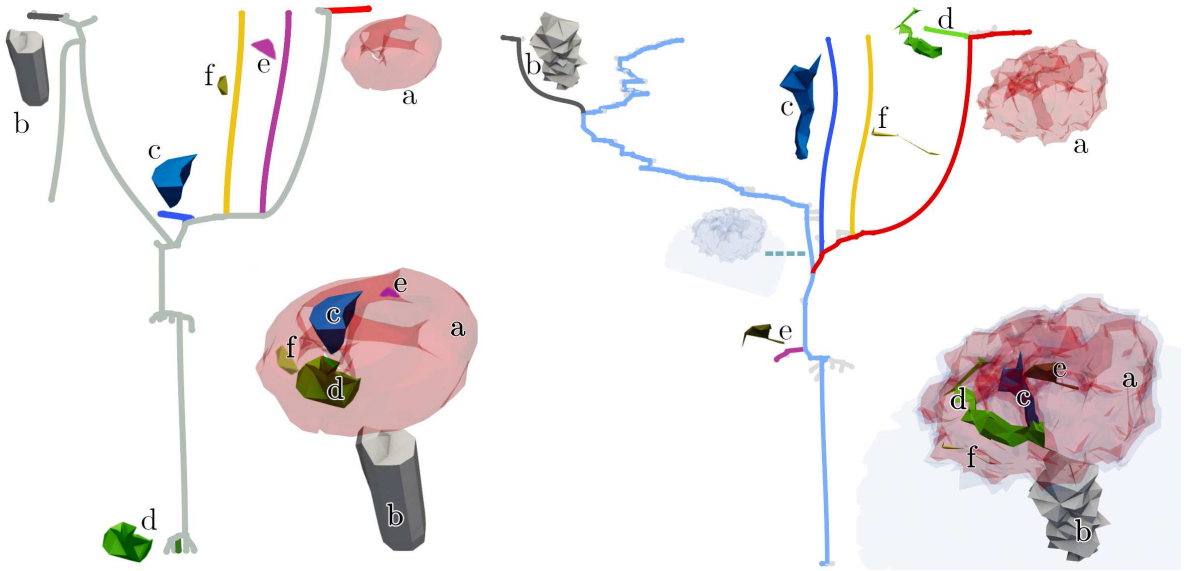


Figure 4: Contour trees computed from the 24^3 grid (left) and $10K$ subsampled particles (right). For each major branch (ranked by volume), flexible isosurface visualizations are shown and color-coded, alongside the full datasets. Light-gray branches denote low-volume features.

8 Implementation

Given the polynomial coefficients in Sec. 6, we can now turn to the implementation details for computing PIC-based contour trees.

The key decision was whether to implement the new exact volume simplification pipeline in `VTK-m` (now called `Viskores`) or `TTK`. As in Sec. 4, `TTK` already has irregular meshes implemented, but does not have the hypersweep framework needed for summing the coefficients described in Sec. 6. `TTK` is integrated to `ParaView`, while `Viskores` form the basis of distributed contour tree computation methods and is faster at large scales [8]. We therefore decided to implement the pipeline in `Viskores`, for which there are five principal changes required:

- Delaunay tetrahedralization
- Replacing grid-based code with mesh-based code
- Implementing polynomial coefficient computation
- Modifications to hypersweep
- Visualization pipeline changes: flexible isosurface adaptation

Delaunay Tetrahedralization: Since Delaunay meshing is not yet in the `Viskores` codebase, we computed a Delaunay tetrahedralization from the parcel centroids externally with `TetGen` [34], as shown in Fig. 1(c), then read the resulting mesh into `Viskores`.

Grid To Mesh: Given prior experience with the contour tree, considerable effort was expended in the original implementation of `Viskores` to ensure that the contour tree algorithm was abstracted rather than built directly on the grid. It was previously implemented by building a `Mesh` class on top of the input data, for 2D and 3D *Freudenthal* simplicial meshes, plus the ability to compute contour trees based on *Marching Cubes* connectivity. Since the contour tree is a valid *topology graph* for computing itself, a `Mesh` class could effectively encapsulate a contour tree as input to the computation, permitting an easy early way to distributed computation.

In all cases, access to the `Mesh` was abstracted, but in two places, there was an assumption of vertex degree less than 32 for all vertices, allowing the use of a bitmask for efficient representation of vertex neighborhoods. This same vertex bound allowed us to run worklets that iterated around each vertex in early stages: for arbitrary-degree vertices, this can degenerate to serial performance.

We therefore implemented a new `Mesh` class to encapsulate an arbitrary `TopologyGraph`, and rewrote the section of code that iterated around a vertex to substitute a PRAM-efficient segmented sort to select the best ascending edge for each vertex. Since the original code was templated on the `Mesh`, it was not difficult to ensure that the existing code path was preserved for grid-based data.

Polynomial Coefficients: As seen in Sec. 6, further code was also required to compute the correct polynomial coefficients and deltas as pre-processing. This was done with a single parallel loop to compute all of the deltas for each cell, then a segmented sort to collect them in segments corresponding to each vertex in the mesh. Finally, a segmented prefix sum computed the total delta at each vertex, which was used as input to the following hypersweep.

Modifications to Hypersweep: The existing hypersweep code assumed a single (integer) value per vertex, and inversion could be easily performed by subtracting from the total volume and adding 1. For full polynomial coefficients, we had to upgrade the Hypersweep to a version templated on a data type (in our case, vectors of four doubles to represent the coefficients) and which correctly handled arbitrary deltas at each supernode.

Visualization Pipeline: Previous work [22] computed the branch decomposition based on node counts, and had to be templated to allow arbitrary data types. We also carried over the flexible isosurface [11] extraction of single contours by first reconstructing entire surfaces at critical points with marching tetrahedra, then choosing only those triangles that mapped to the correct superarc, as before. This is different for the main branch (the one with the largest volume). As it connects to many other branches, any extracted isosurface would either occlude inner structures or be occluded by them. Therefore, we select the middle iso-value, and later use transparency to show it.

As we will see in Sec. 9, these modifications were successful, and we expect to contribute them to `Viskores` in due course.

9 RESULTS

We used an EPIC [20] simulation of a developing cloud at a mature stage, with $\sim$2 million parcels, and its resampled version onto a 1024^3 grid. We evaluated performance, accuracy, and quality, comparing the time cost and memory use of our code to contour tree implementations in `Viskores` and `TTK`. We then examined the contour tree structure and displayed flexible isosurfaces side-by-side.

Hardware: Because contour tree computation has a significant memory footprint (Sec. 5), we chose to run our benchmarks on the Aire HPC Facility. We use the standard compute partition that offers CPU nodes with AMD Dual 84 core 2.2GHz (9634 Genoa-X) processors and 768 GB DDR5-4800 memory.

Software: Our method builds on `VTK-m` 2.3.0 (at the time of its rebranding as `Viskores` 1.0.0). We compare our `Viskores` irregular mesh implementation against `TTK` 1.3.0 (the most recent stable release at the time of our evaluation), and include three contour tree construction baselines: on regular grids (timing, memory Tab. 1), and on irregular meshes (timing Tab. 2, memory Tab. 3). All benchmarks exclude I/O and were run in parallel, on 128 threads.

Data Sets: We use a parcel-based simulation generated by atmospheric scientists for studying cloud formation [20]. The simulation tracks two million parcels and we use the parcel centroids from one of the time steps when the cloud has already matured. We use the *total water specific humidity* scalar field (i.e. *kg* of water vapour plus liquid water per *kg* of air), as the scientists are interested in studying the cloud droplet distribution (Sec. 2).

Methodology (Grid): The original $\sim$2 million parcel mesh-free simulation was first recomputed onto a uniform 1024^3 mesh, following [19] Appendix H, and then coarse-grained to grid resolutions from 24^3 to 512^3. Tab. 1 shows the statistics collected, while Fig. 6 shows flexible isosurface visualizations for all resolutions.

We recorded the data resolution, the number of input points (i.e. regular nodes), and the construction time for respective `TTK` and `Viskores` contour tree filters. For `Viskores`, we also show the time to count regular nodes for volume approximation, as well as the branch decomposition. We later use this to compare our new irregular mesh polynomial hypersweep, which computes *exact* volume weights, and is more complex. `TTK` outperforms `Viskores` for smaller resolutions, while larger resolutions take a significant amount of time ($>$ 16 times slower than `Viskores`). This is not a surprise, since `TTK` targets workstations rather than many-core architectures and relies more on coarse-grain parallelism.

We used `valgrind` to record peak memory usage. At lower resolutions, `Viskores` uses half the memory of `TTK`, but converges for grids with more than $16M$ data points. We suspect this is because `TTK` switches to a fully implicit triangulation for meshes with $> 16,777,216$ vertices. As the data scales, the size per data point ends up around $80-100B$ per sample point in both `Viskores` and `TTK`. At the highest resolution, `Viskores` is at the lower-end of this range ($86B$) compared to `TTK` ($101B$). For the smallest resolutions, the ratio is over 100, but this is likely due to general overhead.

In Fig. 6, we see that the thermal column and cloud body are not reliably visible on grids until 128^3 samples are available, but that a separate component at the top (the *pileus*) is visible throughout. We also see that the vortex ring is hard to make out until higher resolutions, and tends to be visible as multiple components rather than the single toroidal shape we expect.

Methodology (PIC): We computed contour trees and exact volume weights from the EPIC simulation using the Delaunay pipeline (Sec. 8), testing smaller data sizes with random downsamples of the parcels. We show visualizations in Fig. 7 and compare the performance to `TTK` in tables Tab. 2, Tab. 3 and plots Fig. 5.

Tab. 2 shows the time costs of: Delaunay tetrahedralization, contour tree construction, the simplification using exact volume in `Viskores`, and computing the contour tree in `TTK` (join/split trees and the merge). We do not report the simplification time cost in `TTK` because it does not have an exact volume computation.

The Delaunay tetrahedralization is quasilinear [35, 34] and dominates the time cost compared to the contour tree computation in both `Viskores` and `TTK`. However, even when we add all the total time steps: the tetrahedralization, the contour tree construction and (in `Viskores`) the exact volume simplification, the time cost is still smaller than for the highest resolution resampled grids (1024^3 in `Viskores` and both $512^3, 1024^3$ in `TTK`). Moreover, it took $\sim$23 *minutes* to compute the 1024^3 regridding of the original mesh-free simulation in the first place (compared to just 13 seconds for the Delaunay meshing on the original $\sim$2 million parcels).

We also break down the contour tree construction time. Both `Viskores` and `TTK` have two steps, building the edge list for each vertex, and running the contour tree constructor. In `Viskores`, the first step involves a prefix-sum style parallel worklet to extract the edge list from the delaunay mesh, while `TTK` uses a serial `OneSkeleton` extraction step. This is reflected in the timings, where `OneSkeleton` remains the dominant time cost for all resolutions. However, the serial extraction is undoubtedly parallel, so we primarily compare the main step, that of computing the contour tree itself. In `Viskores`, we modified the existing `ContourTreeAugmented` worklet to handle arbitrary graph input (to support arbitrary vertex neighbourhood degrees), causing a significant slow-down compared to the original `Viskores` grid version and to `TTK`, as we can no longer rely on the optimisation that assumed < 32 maximum neighbours per vertex (Sec. 8).

We note that the ratio of the `Viskores CTA` / `TTK FTMTree` filter timings is trending downwards with bigger datasets Fig. 5(a) in much the same way as for grids. We predicted that around 30M parcels the `Viskores CTA` worklet would break even with the `FTMTree` filter. We ran a comparison for a larger, different, 30M parcel simulation, and measured that `Viskores/CTA` contour tree construction was twice as fast as `TTK/FTMTree` (Fig. 5(a) - cross).

We then turn our attention to the time cost of exact volume computation in `Viskores`. Compared to grid-based volume approximation, this requires accessing the tetrahedral mesh during the hypersweep, and is in the region of $40-150$ times slower for the same number of regular nodes. This is because collecting the coefficients requires accessing the vertex coordinates and performing millions of double floating-point operations. While the computation of volume weights dominates the time cost, the branch decomposition occupies only a small fraction of the cost but is still about $3-4$ times slower than for the same number of input points on the grid. This is caused by the contour tree having an order of 10 times more branches when computed from our EPIC Delaunay meshing. Computation of volumetric weights is more expensive per vertex than for grids, but since we need to sum multiple sets of polynomial coefficients rather than just use the value 1, this is not a surprise.

For memory usage, Tab. 3 lists the number of sample parcels, the number of tetrahedra in the Delaunay mesh, the peak memory usage according to `valgrind` for both `Viskores` and `TTK`, and the overall cost per data point. For `Viskores`, we also report more fine-grained memory information, such as the memory for raw data and vertex positions, contour tree construction, and simplification. Unsurprisingly, an explicit tetrahedral mesh requires more memory than an implicit gridded mesh with the same number of data points (Tab. 1), but the memory footprint is around $1000B$ overall per data point, and the highest resolution (all $2.1M$ data points) occupies only $2GiB$ of memory, where the largest (1024^3) gridded data is $96GiB$. In `TTK`, the memory footprint is around $4000B$ per data point for smaller resolutions, and $6000B$ for the largest, compared to $543B$ in `Viskores`. In Fig. 5(b), we see that the `Viskores` / `TTK` ratio of memory footprint (for just the contour tree construction) is slightly trending downwards, consistently remaining below 1.

Input		Timing						Memory Usage				
		Viskores (VTK-m)				TTK		Viskores (VTK-m)			TTK	
		... of which						... of which				
Resolution (cubed)	Input Size	Total Time (seconds)	Construction	Weights	Branch Decomposition	Total Time (seconds)	Time Ratio (Viskores / TTK)	Peak Memory (MiB)	Raw Data (MiB)	Bytes per Data Point	Peak Memory (MiB)	Bytes per Data Point
24	14K	0.068	0.0662	0.0003	0.0019	0.003	22.08	1.6	0.053	111	3.232	234
32	32K	0.073	0.0708	0.0005	0.0021	0.006	11.80	3.48	0.125	102	6.713	205
48	110K	0.120	0.1162	0.0011	0.0024	0.021	5.536	11.19	0.422	97	21.07	191
64	262K	0.137	0.1319	0.0022	0.0028	0.040	3.298	26.29	1	97	48.93	187
96	880K	0.167	0.1571	0.0066	0.0031	0.120	1.310	88.09	3.375	96	163.8	185
128	2.1M	0.241	0.2226	0.0151	0.0034	0.254	0.877	208.5	8	95	395.3	188
192	7M	0.482	0.4244	0.0510	0.0071	0.903	0.470	703.3	27	96	1,296	183
256	16M	0.895	0.7636	0.1204	0.0106	2.623	0.291	1,628	64	93	3,103	185
384	57M	2.285	1.8687	0.4032	0.0132	11.730	0.159	5,489	216	93	5,953	105
512	134M	4.674	3.5602	0.9544	0.1590	34.874	0.102	13,010	512	93	14,030	105
1024	1B	37.574	29.5676	7.7865	0.2195	504.800	0.059	96,700	4096	86	108,594	101

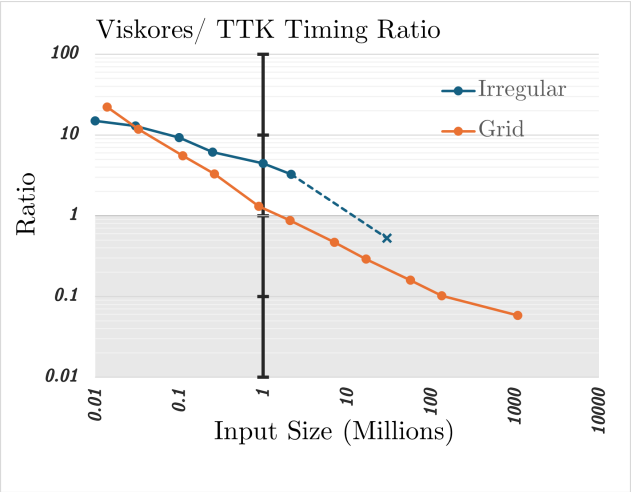
Table 1: Comparison of *timings and memory usage* for `Viskores` (`VTK-m`) and `TTK` for grid-based data. We also note that it takes ∼23 minutes just to grid the original simulation onto the 1024^3 mesh. The resolutions 24^3 to 512^3 are down-sampled versions of the 1024^3 grid.

Num. of Points	Pre-Processing	Viskores (VTK-m)						TTK			Ratio
		... of which			... of which			... of which			
	Delaunay (seconds)	Construction Total Time (seconds)	Traversal	CTA Worklet	Simplfication Total Time (seconds)	Volume Weights	Branch Decomposition	Construction Total Time (seconds)	Traversal (OneSkeleton)	FTMTree Filter	Time Ratio (CTA / FTMTree)
10K	0.040	0.142	0.022	0.120	0.020	0.014	0.006	0.019	0.011	0.008	15.01
30K	0.168	0.210	0.055	0.155	0.044	0.037	0.007	0.045	0.033	0.012	12.93
100K	0.425	0.329	0.079	0.250	0.113	0.105	0.007	0.125	0.098	0.027	9.24
250K	1.455	0.499	0.156	0.343	0.238	0.230	0.008	0.312	0.256	0.056	6.13
1M	5.470	1.214	0.331	0.883	1.049	1.038	0.011	1.245	1.046	0.199	4.44
2.1M	13.889	2.075	0.646	1.429	2.155	2.140	0.016	2.603	2.162	0.441	3.24
30M*	225.84*	42.39*	7.72*	34.671*	71.758*	70.998*	0.759*	102.60*	36.86*	65.743*	0.53*

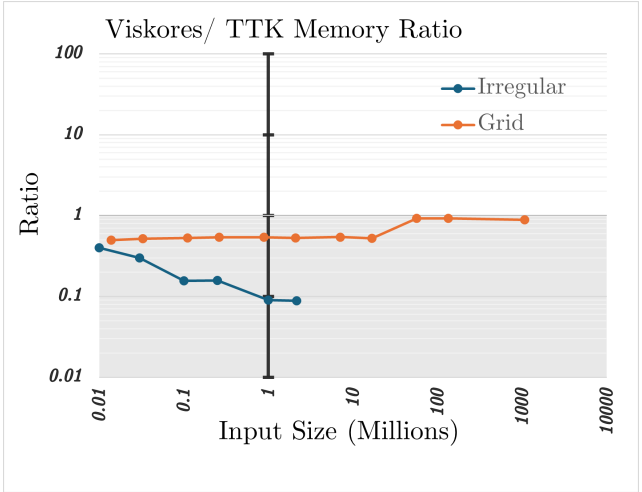
Table 2: Comparison of contour tree construction *timings* on an irregular mesh in `Viskores` (`VTK-m`) and `TTK`.
Corresponding filters are: `CTA` (`ContourTreeAugmented`) in `Viskores`, and `FTMTree` (*Fibonacci Task-based Merge tree)* in `TTK`.
Row marked with '*' comes from a different, 30M parcel simulation for time estimation on larger data.

Input		Viskores (VTK-m)							TTK		Ratio
		... of which			Bytes per data point						
Num. of Points	Tetrahedra	Peak Memory (MiB)	Raw Data	Construction	Simplification	Construction	Simplification	Total	Peak Memory (MiB)	Bytes per Data Point	Construction Memory Ratio (Viskores/TTK)
10K	67K	21	0.23	16.10	4.7	1,610	490	2,100	40.29	4029	0.399
30K	201K	50.7	0.69	35.92	14.1	1,197	493	1,690	120.4	4013	0.298
100K	671K	111.6	2.29	62.28	47.0	623	493	1,116	400.5	4005	0.155
250K	1.7M	277.1	5.72	153.78	117.6	615	493	1,108	978	3912	0.157
1M	6.3M	1,029	22.88	558.40	447.7	558	471	1,029	6,203	6203	0.090
2.1M	13M	2,160	49.44	1174.45	936.1	543	456	1,000	13,240	6127	0.089

Table 3: Comparison of *memory usage* associated with contour tree analysis in `Viskores` (`VTK-m`) and `TTK` for an irregular mesh.



(a) Time usage trend comparison. Data from Tab. 1, Tab. 2



(b) Memory usage trend comparison. Data from Tab. 1, Tab. 3

Figure 5: Logarithmic plots showing the $\frac{\texttt{Viskores}}{\texttt{TTK}}$ ratios in contour tree construction *timings* and *memory use*.
The x-axis shows the input sizes (millions of vertices); y-axis shows the ratios, with the shaded region below $y = 1$ indicating better `Viskores` performance. Blue lines are measures on irregular data: (a) Tab. 2 (Time Ratio `CTA/FTMTree`), (b) Tab. 3 (Bytes per Data Point: `Viskores` (Construction) / `TTK`). Dashed blue line with the cross marker in (a) is from a larger, different (30M) simulation. Orange lines indicate grid data measures: (a) Tab. 1 (Time Ratio `CTA/FTMTree`), (b) Tab. 1 (Bytes per Data Point: `Viskores` / `TTK`).

Comparison: Our original hypothesis was twofold: that contour tree analysis was more efficient and accurate without having to resample irregular meshes, and that flexible isosurface visualizations were potentially useful in studying cloud formation.

Looking at Fig. 4, the contour tree structures appear similar, except that we see more branches in the tree generated directly from the parcels because critical points are guaranteed to appear at the vertices [6]. The color-coded branches (which represent the largest features by volume) are labelled *a-f* by order of volume and appear in similar regions in both datasets. The biggest features *a, b, c* can be easily matched to well-known parts of a cloud, namely the vortex ring, the thermal column, and the pileus (the cap), in that order; *d, e, f* are more difficult to pinpoint, especially since (*d*, *e*) appear as both topological peaks or pits, but look to be part of the vortex core - part of the cloud typically associated with strongest vorticity.

As we see in higher resolutions (Fig. 6, Fig. 7) we can reliably discern features of the cloud. In Fig. 6 and Fig. 7 we can see the quality improving as the resolution increases. However, one effect that we had not anticipated was the breaking-up of the vortex ring in gridded resampling, with the direct parcel-based version producing more consistent results. We suspect that this is a side-effect of the interpolants used in the resampling, but were unable to test this.

In terms of efficiency, we can see that the direct computation with $2.1M$ parcels is time-competitive with gridded resamplings at around 384^3, and memory-competitive at around 256^3 but that in the visualizations, the rising cloud stem and the vortex ring starts being visible even with as few as $10K$ parcels, and the $1M$ and $2M$ parcel versions appear to show not only the vortex ring itself but also the vortex core. By the time that a regridded resolution starts showing the vortex core and ring reliably, however, the compute time is $2\times$ slower and the memory footprint $50\times$ greater. We also note that resampling $\sim$2 million parcels onto a 1024^3 grid can take up to 30 *minutes* even when using task-based parallelism (when using parcel resampling methods described in [19] Appendix H), whereas the Delaunay tetrahedralization only takes about 13 seconds. Additionally, by doing the topological analysis directly on the parcels, the interpolation artifacts are avoided, such as the smoothing and blurring of the features; however, further studies on alternative neighbourhoods than just Delaunay are also of great interest. The downside of the coefficient-style exact volume computations is the summation of many very small floating point numbers, which is likely to cause precision errors for larger data, as we already noticed the dynamic range of the summation sequence reaching 1e+13, prompting more future research for applying computer arithmetic methods for better numerical stability (especially in-parallel).

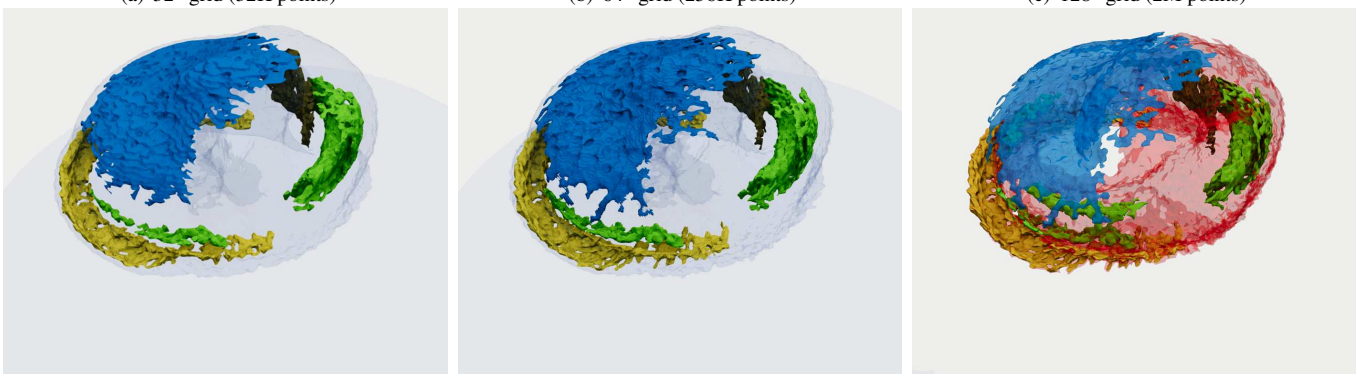
We would therefore conclude that, where possible, both topological analysis and regular visualization of EPIC simulations is best done with the original parcels, after a meshing them with a Delaunay tetrahedralization rather than by resampling to a grid just to take advantage of existing tools.

10 Summary and Future Work

We have reported correctly developed polynomial coefficients for geometric properties in the well-known Contour Spectrum [1], and demonstrated how to compute them in PRAM parallel for data on an irregular mesh for contour tree analysis. This enables the computation of exact (i.e. not approximated) volumes for important regions of parcel-based scientific simulations. We have also reported on the implementation changes in the `Viskores` library necessary to add irregular mesh analysis to the existing methods. We have shown that for a known data set, working directly with the underlying data representation is preferable to resampling just for the purpose of data visualization, and have shown that the efficiency gains in doing so can be several orders of magnitude.

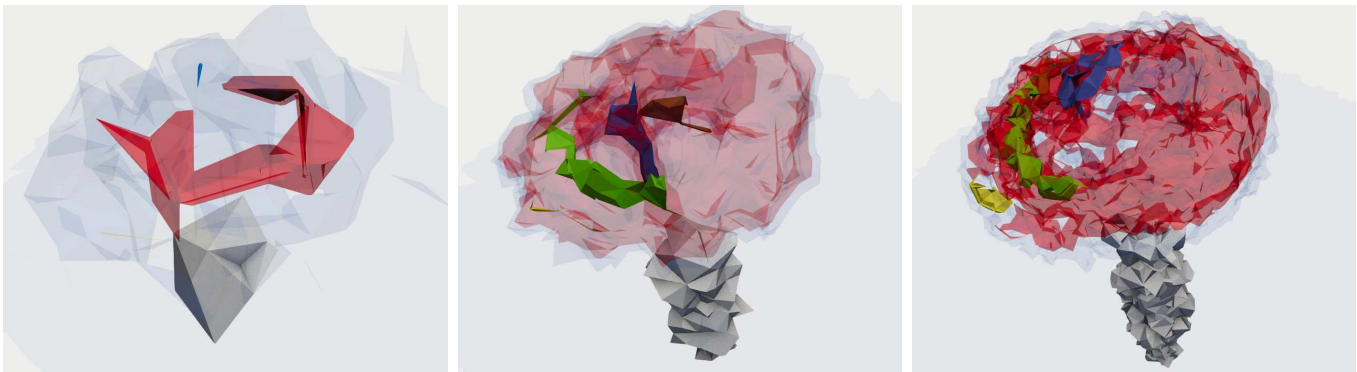
Several lines of inquiry now open up, in particular to return to Pascucci's computation of Betti numbers [28], but also to expand the hypersweep computation to a broader range of geometric measures [11], now that the basic mechanisms have been tested. Some more care will be needed to handle the numerical precision of this technique, which can become the limiting factor if not addressed. Inevitably, though, this work will also need porting to the distributed layer, as the real value of topological analysis is to be found at the supercomputer scale rather than the single node which we used in our experiments.

(a) 32^3 grid (32K points) (b) 64^3 grid (256K points) (c) 128^3 grid (2M points)

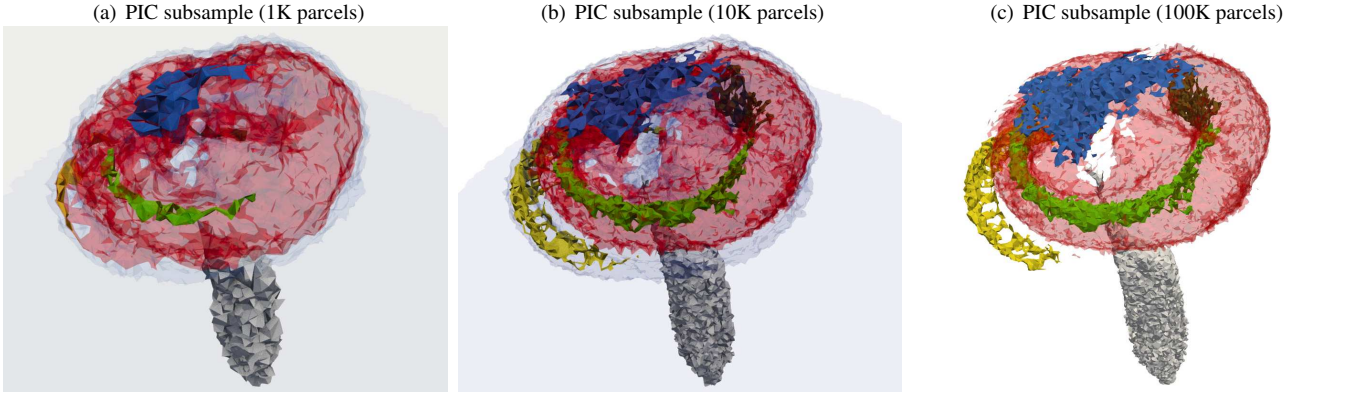


(d) 256^3 grid (16M points) (e) 512^3 grid (128M points) (f) 1024^3 grid (1B points)

Figure 6: Grid-based visualizations at different resolutions.



(a) PIC subsample (1K parcels) (b) PIC subsample (10K parcels) (c) PIC subsample (100K parcels)



(d) PIC subsample (200K parcels) (e) PIC subsample (1M parcels) (f) PIC - the original 2.1M parcels

Figure 7: PIC-based visualizations based on random sub-selection of parcels.

Acknowledgments

We thank the University of Leeds and UKRI for supporting the research (Grants: EP/W524372/1 DTP Studentship 2751073 and EP/T025409/1). This work was undertaken on the Aire HPC system at the University of Leeds, UK, with the simulation data produced on ARCHER2 UK National Supercomputing Service [2]. For technical assistance with flexible isosurfaces, we thank Petar Hristov.

References

[1] C. L. Bajaj, V. Pascucci, and D. R. Schikore. The Contour Spectrum. In *Proceedings of Visualization 1997*, pp. 167–173, 1997. doi: 10.1145/259081.259279 1, 3, 8

[2] G. Beckett, J. Beech-Brandt, K. Leach, Z. Payne, A. Simpson, L. Smith, A. Turner, and A. Whiting. ARCHER2 Service Description, Dec. 2024. doi: 10.5281/zenodo.14507040 10

[3] A. Blyth, T. Choularton, G. Fullarton, J. Latham, C. Mill, M. Smith, and I. Stromberg. The Influence of Entrainment on the Evolution of Cloud Droplet Spectra .2. Field Experiments at Great-Dun-Fell. *Quarterly Journal of the Royal Meteorological Society*, 106(450):821–840, 1980. doi: 10.1002/qj.49710645012 2

[4] A. M. Blyth. Entrainment in Cumulus Clouds. *Journal of Applied Meteorology and Climatology*, 32(4):626–641, 1993. doi: 10.1175/1520-0450(1993)032<0626:EICC>2.0.CO;2 2

[5] S. J. Böing, D. G. Dritschel, D. J. Parker, and A. M. Blyth. Comparison of the Moist Parcel-in-Cell (MPIC) model with large-eddy simulation for an idealized cloud. *Quarterly Journal of the Royal Meteorological Society*, 145(722):1865–1881, 2019. doi: 10.1002/qj.3532 2

[6] H. A. Carr. *Topological Manipulation of Isosurfaces*. PhD thesis, University of British Columbia, 2004. doi: 10.14288/1.0051287 1, 8

[7] H. A. Carr, O. Rübel, and G. H. Weber. Distributed Hierarchical Contour Trees. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pp. 1–10, 2022. doi: 10.1109/LDAV57265.2022.9966394 3

[8] H. A. Carr, O. Rübel, and G. H. Weber. Distributed Hierarchical Contour Trees. In *2022 IEEE 12th Symposium on Large Data Analysis and Visualization (LDAV)*, pp. 1–10. IEEE, 2022. doi: 10.1109/LDAV57265.2022.9966394 5

[9] H. A. Carr and J. Snoeyink. Representing Interpolant Topology for Contour Tree Computation. In H.-C. Hege, K. Polthier, and G. Scheuermann, eds., *Topology-Based Methods in Visualization II*, Mathematics and Visualization, pp. 59–74. Springer, 2009. doi: 10.1007/978-3-540-88606-8_5 3

[10] H. A. Carr, J. Snoeyink, and M. van de Panne. Simplifying Flexible Isosurfaces Using Local Geometric Measures. In *IEEE Visualization 2004*, pp. 497–504, 2004. doi: 10.1109/VISUAL.2004.96 1

[11] H. A. Carr, J. Snoeyink, and M. van de Panne. Flexible Isosurfaces: Simplifying and Displaying Scalar Topology Using the Contour Tree. *Computational Geometry: Theory and Applications (CGTA)*, 43(1):42–58, 2010. doi: 10.1016/j.comgeo.2006.05.009 2, 3, 5, 8

[12] H. A. Carr, G. H. Weber, C. Sewell, and J. Ahrens. Parallel Peak Pruning for Scalable SMP Contour Tree Computation. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pp. 75–84, 2016. doi: 10.1109/LDAV.2016.7874312 3

[13] C. Correa and P. Lindstrom. Towards robust topology of sparsely sampled data. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1852–1861, 2011. doi: 10.1109/TVCG.2011.245 3

[14] J. O. Dabiri and M. Gharib. Fluid entrainment by isolated vortex rings. *Journal of fluid mechanics*, 511:311–331, 2004. doi: 10.1017/S0022112004009784 2

[15] J. W. Deardorff. A three-dimensional numerical investigation of the idealized planetary boundary layer. *Geophysical and Astrophysical Fluid Dynamics*, 1(3-4):377–410, 1970. doi: 10.1080/03091927009365780 2

[16] R. A. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '88, pp. 65–74. Association for Computing Machinery, New York, NY, USA, 1988. doi: 10.1145/54852.378484 2

[17] B. Duffy, H. A. Carr, and T. Möller. Integrating Isosurface Statistics and Histograms. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 19(2):263–277, 2012. doi: 10.1109/TVCG.2012.118 4

[18] H. Federer. *Geometric Measure Theory*. Springer-Verlag, 1965. doi: 10.1007/978-3-642-62010-2 3, 4

[19] M. Frey, D. Dritschel, and S. Böing. EPIC: the Elliptical Parcel-In-Cell method. *Journal of Computational Physics: X*, 14:100109, 2022. doi: 10.1016/j.jcpx.2022.100109 2, 6, 8

[20] M. Frey, D. Dritschel, and S. Böing. The 3D Elliptical Parcel-In-Cell (EPIC) method. *Journal of Computational Physics: X*, 17:100136, 2023. doi: 10.1016/j.jcpx.2023.100136 2, 6

[21] C. Gueunet, P. Fortin, and J. Jomier. Contour forests: Fast multi-threaded augmented contour trees. In *6th IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pp. 85–92, Oct 2016. doi: 10.1109/LDAV.2016.7874333 3

[22] P. Hristov, G. H. Weber, H. A. Carr, O. Rübel, and J. Ahrens. Data Parallel Hypersweeps for in Situ Topological Analysis. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pp. 12–21, 2020. doi: 10.1109/LDAV51489.2020.00008 3, 4, 5

[23] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, 1988. doi: 10.1109/38.511 2

[24] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH Computer Graphics*, 21(4):163–169, 1987. doi: 10.1145/37402.37422 2

[25] J. P. Mellado. Cloud-Top Entrainment in Stratocumulus Clouds. *Annual Review of Fluid Mechanics*, 49:145–169, 2017. doi: 10.1146/annurev-fluid-010816-060231 2

[26] G. L. Miller and J. H. Reif. Parallel Tree Contraction and Its Application. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, vol. 26, pp. 478–489, 1985. doi: 10.1109/SFCS.1985.43 3

[27] K. Moreland, C. Sewell, W. Usher, L.-T. Lo, J. Meredith, D. Pugmire, J. Kress, H. Schroots, K. L. Ma, H. Childs, M. Larsen, C. M. Chen, R. Maynard, and B. Geveci. VTK-m: Accelerating the Visualization Toolkit for Massively Threaded Architectures. *IEEE Computer Graphics and Applications*, 36(3):48–58, May 2016. doi: 10.1109/MCG.2016.48 1, 3

[28] V. Pascucci. On the Topology of the Level Sets of a Scalar Field. In *Abstracts of the 13th Canadian Conference on Computational Geometry*, pp. 141–144, 2001. 3, 8

[29] V. Pascucci and K. Cole-McLaughlin. Parallel Computation of the Topology of Level Sets. *Algorithmica*, 38(2):249–268, 2003. doi: 10.1007/s00453-003-1052-3 3

[30] V. Pascucci, K. Cole-McLaughlin, and G. Scorzell. Multi-Resolution computation and presentation of Contour Trees. In *Proceedings of the IASTED conference on Visualization, Imaging and Image Processing (VIIP 2004)*, pp. 452–290, 2004. 3

[31] G. Reeb. Sur les points singuliers d'une forme de Pfaff complètement intégrable ou d'une fonction numérique. *Comptes Rendus de l'Acadèmie des Sciences de Paris*, 222:847–849, 1946. doi: 10.24033/asens.776 2

[32] C. E. Scheidegger, J. M. Schreiner, B. Duffy, H. A. Carr, and C. T. Silva. Revisiting Histograms and Isosurface Statistics. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1659–1666, 2008. doi: 10.1109/TVCG.2008.160 3

[33] C. E. Shannon. Communication in the Presence of Noise. *Proceedings of the IRE*, 37(1):10–21, 1949. doi: 10.1109/JRPROC.1949.232969 1, 3

[34] H. Si. TetGen A Delaunay-Based Quality Tetrahedral Mesh Generator. *ACM Trans. Math. Softw*, 41(2):11, 2015. doi: 10.13140/RG.2.2.13915.85284/2 5, 6

[35] H. Si. *TetGen A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator*, 2020. 6

[36] J. Tierny, G. Favelier, J. A. Levine, C. Gueunet, and M. Michaux. The Topology ToolKit. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):832–842, 2017. doi: 10.48550/arXiv.1805.09110 3

[37] B. Zhou, Y.-J. Chiang, and C. Wang. Efficient Local Statistical Analysis via Point-Wise Histograms in Tetrahedral Meshes and Curvilinear Grids. *IEEE Transactions on Visualization and Computer Graphics*, 25(2):1392–1406, 2018. doi: 10.1109/TVCG.2018.2796555 1, 3