Compression and Ray-March Rendering using Implicit Neural Representations of Data Values and Domain Extent

Robert Sales * and Graham Pullan †
Whittle Laboratory, Department of Engineering, University of Cambridge, Cambridge, UK

Unlike traditional approaches, neural representations can compress unstructured data from aerospace simulations by factors of 200× while retaining sufficient numerical quality for visualisation. These work by approximating the underlying field, mapping coordinates to values via compact neural networks. A key practical limitation, however, is that this mapping does not encode the spatial extent of the original dataset domain, which almost always leads to erroneously extrapolated artefacts in rendered images produced by implicit ray-march rendering. Previous compression studies avoid this by separately storing the original coordinates to reconstruct an explicit version of the data for post-compression visualisation. This approach, however, reduces the effective compression of both coordinates and values to a negligible amount. As a solution, we propose a dual-network framework that jointly encodes both data values and domain extent within a fully implicit, compressive neural representation. Our approach uses a compressed neural signed distance function to constrain data to its true bounds, enabling implicit rendering without intermediate decompression. We validate this approach using our hybrid sphere-tracing and ray-marching method for isosurface rendering, and demonstrate its effectiveness — along with several optimisations — using a CUDA proof-of-concept viewer.

I. Introduction

Advances in hardware and high-performance parallel computing continue to enable increasingly sophisticated numerical simulations, generating ever-larger volumetric datasets. High-resolution models in domains such as fluid dynamics and atmospheric science routinely produce tens of gigabytes per snapshot [1, 2]. As simulation fidelity rises to exploit available computational capacity, the resulting data volumes pose a persistent and increasingly difficult storage challenge. Traditional lossy compression algorithms can alleviate this problem, but are struggling to keep pace with the growth in dataset size, particularly when high fidelity must be preserved at extreme compression.

Neural representations have recently emerged as a highly effective means of compressing volumetric data. Recent approaches achieve lossy compression of scalar field values from fluid dynamics simulations at factors exceeding 250× with minimal perceptible visual degradation [3]. This is accomplished by learning an implicit approximation of the underlying mapping from coordinates to scalar values via a neural network parameterised by trainable weights and biases. Compression is achieved by constraining the network capacity to be smaller than the number of dataset samples by a factor γ, known as the compression ratio. Implicit representations are particularly well-suited for rendering methods such as ray-marching because they inherently provide sampling at arbitrary resolution.

A key limitation for rendering, however, is that implicit representations do not encode the spatial extent of the original domain and therefore cannot determine whether queried coordinates (along rays) lie inside or outside the true bounds. This matters because implicit functions produce outputs for all inputs — including those beyond the range observed during training — while only points within the domain should appear in visualisations. Additional mechanisms are therefore required to verify queried coordinates before decompressed values can be considered. Without validation, erroneously extrapolated artefacts often appear and obscure true data, as shown in Fig. 1.

An explicit reconstruction of the target dataset can be produced by querying the neural representation at original coordinates, which must be stored separately and almost entirely negates the benefits of neural representations. Currently available alternatives suitable for implicit rendering instead store either volume representations (e.g., voxel grids or point clouds) or boundary representations (e.g., triangular surface meshes), neither of which are themself implicit or

*PhD Student.
†Professor of Computational Aerothermal Design, AIAA Senior Member.

particularly well-suited to extreme compression. As a result, the overall compression factor considering both data values and domain is far smaller than that achievable for the data values alone.

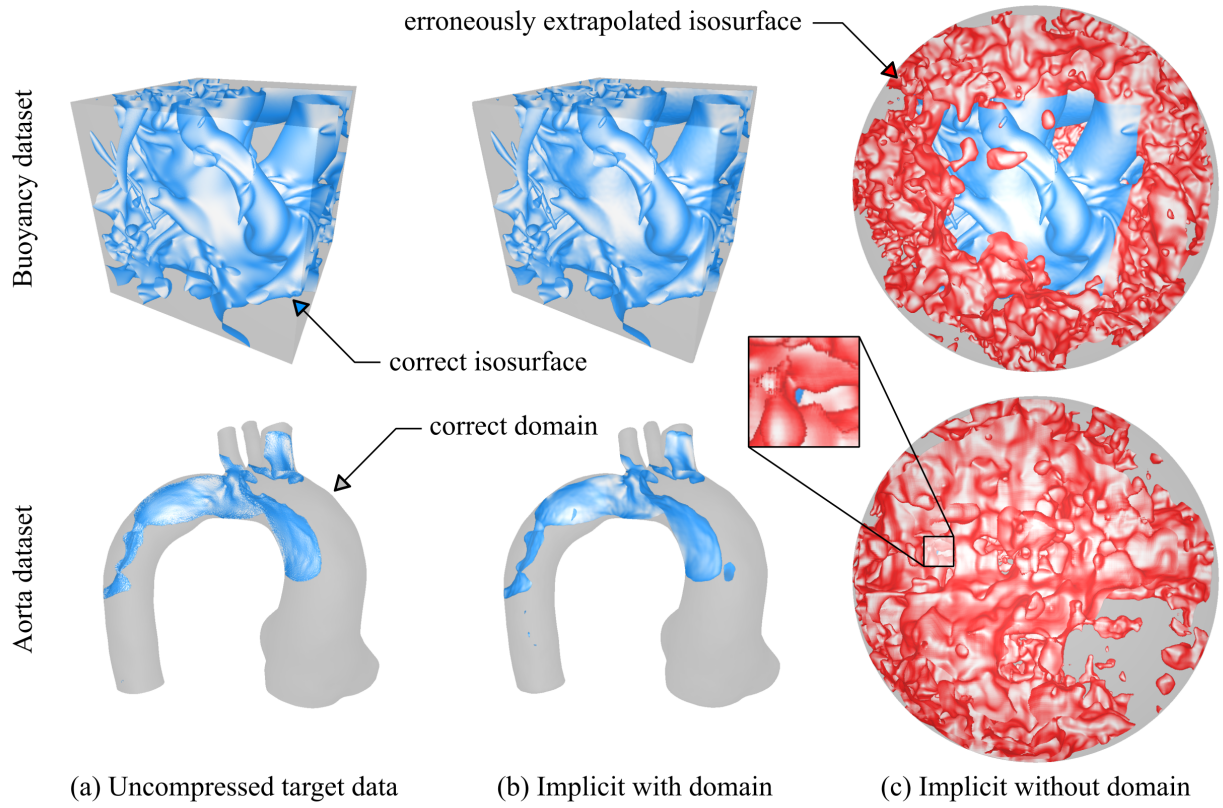

Fig. 1 Isosurfaces produced using ray-march rendering on (a) the target data, (b) 25× neural compressed values with neural compressed domain, and (c) 25× neural compressed values with bounding sphere. Erroneously extrapolated isosurfaces outside the target domain, highlighted in red, demonstrate the need for accurate domain representation to avoid artefacts when ray-march rendering values from compressed neural representations.

In this paper, we introduce a dual-network, fully implicit, compressive framework that encodes both data values and domain extent, to address the need for a compressed domain representation and enable implicit rendering without intermediate decompression. In particular, we focus our demonstrations on isosurface (contour) rendering because this is a common yet challenging task. The key contributions of this paper include:

1) A fully-implicit, dual-neural-network framework that constrains compressed representations of volumetric data values within the original domain extent. This is achieved with a compressed neural signed distance function (SDF) that encodes both domain shape and position.
2) A hybrid sphere-tracing and ray-marching approach for rendering isosurfaces from neural-compressed volumes. It is trivial to show that this approach can be extended to a range of surface and volume rendering applications that are popular in data visualisation, such as cut-planes or multi-contours.
3) A level-of-detail (LoD) cascade and Lipschitz-informed adaptive marching regime that significantly reduce the number of ray steps per image, enabling higher frame rates while preserving image quality. This is demonstrated in a proof-of-concept CUDA viewer that we publish alongside this paper.

The remainder of this paper is structured as follows: Section II reviews prior work on compression techniques for representing data values and for representing domain shape. Section III introduces the dual-neural-network framework and describes the compressed representation of domain shape via a neural signed distance function. It also introduces the rendering implementation, with Lipschitz adaptive marching and a level-of-detail cascade optimisations. Section IV presents comparative results versus traditional compressors, evaluates the proposed optimisations, and benchmarks the CUDA-based renderer. Finally, section V summarises the contributions and outlines directions for future work.

II. Previous Work

This section summarises previous work on the compression of volumetric field values (Section II.A) and three-dimensional shapes (Section II.B). Because accurate rendering depends on both components, any meaningful reduction in overall storage footprint requires that the domain representation be compressed with efficiency comparable to that of the field data itself. We omit a discussion of scene representation networks, such as NeRFs [4–6] and Gaussian Splatting [7–9], since these are designed to synthesize photorealistic images rather than encode actual data values.

A. Compressing Volumetric Values

Lossless compression achieves minimal reductions in data size on high-entropy scientific data, making lossy compression essential for storing and visualising large simulation datasets [10]. Such techniques generally fall into three categories: explicit transform methods, explicit predictive methods, and neural implicit methods that leverage deep learning.

Explicit transform methods apply mathematical transforms to decompose data into components associated with distinct frequency bands, achieving compression by quantising or discarding low-energy components and entropy-encoding the remaining coefficients. The Discrete Cosine Transform (DCT) — which is central to JPEG [11] and MPEG standards [12] — concentrates most signal energy into low-frequency coefficients and is highly effective for smoothly varying image data. The Discrete Fourier Transform (DFT) similarly decomposes data into global sine and cosine modes and is best suited to smooth, periodic signals; however, both DCT- and DFT-based methods perform poorly on non-smooth or noisy data and exhibit Gibbs phenomenon ringing near discontinuities. Wavelet transforms, such as those of Haar [13] and Daubechies [14], provide joint spatial–frequency localisation, improving compression of aperiodic data with sharp features [15]. More recent extensions to multidimensional scientific data include ZFP, which applies an orthogonal block transform and embedded coding to 4^d blocks under a fixed-bit budget [16], and TTHRESH, which uses higher-order singular value decomposition followed by bit-plane, arithmetic, and run-length encoding [17]. MGARD performs a multilevel finite-element–style decomposition followed by coefficient quantisation and entropy coding [18], while SPERR applies a multiresolution wavelet transform with SPECK-style embedded coding and an outlier-refinement stage to achieve high rate–distortion performance on 2D and 3D floating-point fields [19]. While transform-based methods generally achieve the best accuracy for a given compression ratio, they are restricted to array-based data on structured (i, j, k) grids, which excludes a large proportion of aerospace and turbomachinery datasets defined on unstructured meshes. Although some techniques can be applied to vectorised data, doing so destroys spatial locality and leads to severe degradation in compression performance.

Implicit predictive methods approximate data using low-order mathematical models, applied either to flattened arrays or directly in higher dimensions, storing only the model parameters rather than explicit values. Compression is achieved by limiting the number of parameters and minimising residuals, often followed by parameter quantisation or entropy encoding. Well-known examples include FPZIP [20], which uses the Lorenzo predictor to estimate each point from nearby values, storing only the prediction residuals; these residuals are mapped to integers, grouped, and compressed with range coding, with lossy compression achieved by discarding lower-magnitude bits during the integer conversion process. ISABELA sorts flattened data into monotonic order and fits (by regression) a B-spline curve to approximate values at indexed locations [10]; the pre-sorting reduces the number of spline coefficients required to achieve a given accuracy bound on noisy data but necessitates storing a large inverse permutation vector, limiting maximum compression to $\gamma \approx 10$. SZ [21] and SZ2 [22] similarly flatten blocks of floating-point data, select the best-fit model between consecutive points, and encode it using a compact two-bit scheme: 00_2 for the preceding neighbour, 01_2 for linear curve, 10_2 for quadratic curve, and 11_2 for "unpredictable" points, which are entropy-encoded separately. SZ and its successors can achieve compression ratios of up to 400:1 with high numerical accuracy, though performance is more data-dependent than transform schemes, particularly on unstructured data where consecutive samples are not spatially local and thus weakly correlated. Despite sometimes requiring additional connectivity information, predictive schemes are generally more flexible and can be applied to both structured and unstructured datasets.

Implicit neural methods treat volumetric data as discrete and independent coordinate–value samples, each pairing a spatial coordinate $\vec{x} = (x, y, z)$ with a scalar value v, with no assumed neighbourhood structure or spatial connectivity. These samples are interpreted as observations of an underlying continuous volumetric field $f : \mathbb{R}^3 \to \mathbb{R}$ that, for real data, is far too complex to express analytically. Instead, neural representations exploit the universal approximation capability of multilayer perceptrons (MLPs) [23] to learn this coordinate-to-value mapping via gradient-descent optimisation. Neural representation replaces explicit storage of field values with a neural network whose parameters encode a

continuous approximation of the underlying field. Like classical implicit predictors, neural methods operate effectively on both structured and unstructured data, but additionally provide built-in spatial interpolation for queries at arbitrary resolution, which is a particular advantage for sampling-heavy, implicit ray-marching rendering tasks.

Before modern deep learning, early neural representations used single-hidden-layer radial basis function networks, which took continuous coordinates as input and produced scalar outputs for nonlinear interpolation of scattered samples [24]. This line of work was later extended using multi-layer perceptrons (MLPs) with sigmoidal activations, culminating in universal approximation results demonstrating that such networks can represent arbitrary continuous multidimensional functions [23, 25, 26]. Poggio and Girosi further formalised the problem of learning continuous functions from sparsely sampled data, establishing a theoretical foundation for neural function representation [27]. More direct predecessors to modern neural fields apply coordinate-MLPs with ReLU to represent three-dimensional scenes, trained via stochastic gradient descent for neural rendering applications [28]. Compared to other activations, however, ReLU produces piecewise-linear mappings that limit the network's ability to capture fine detail and higher-order derivatives in complex data. Fourier features [29] address this limitation by projecting low-dimensional coordinates into a higher-dimensional space via log-linearly spaced sine and cosine embeddings, substantially improving a network's ability to capture high-frequency variation. Sitzmann et al. [30] further advance this direction with sinusoidal representation networks (SIRENs), whose periodic activations provide a strong inductive bias for representing complex signals and their derivatives, and consistently outperform ReLU-MLPs, even when augmented with Fourier features.

Lu et al. [3] extend deep SIRENs by adding residual connections, as in ResNet [31], within their *NeurComp* neural compression framework for volumetric scalar fields. They formally define neural compression by constraining the number of network parameters k to be smaller than the number of original samples S, giving a compression ratio $\gamma = S/k$. NeurComp provides fine-grained control over the compression–accuracy trade-off and, with additional post-training parameter quantisation, consistently outperforms TTHRESH [17] — achieving extreme compression ratios of up to 1300:1 with minimal visual error across diverse datasets. Motivated by these results, we adopt a deep SIREN architecture with residual connections and Fourier features for all data-value-compression components of our pipeline. More advanced neural representations have also been proposed: hypernetwork-based approaches such as NIF [32] achieve improved representation efficiency over deep SIRENs, while adaptive domain-partitioning methods, including ACORN [33], SCI [34], and TINC [35], use hierarchical spatial decompositions to allocate capacity more effectively in statistically inhomogeneous domains. These architectures, however, introduce substantial additional constraints on the data format, making them hard or impractical to integrate into our domain-agnostic neural compression and rendering pipeline. Incorporating such methods is beyond the scope of this study but remains promising future work.

B. Compressing Domain Shape

Rendering neural implicit functions requires an additional coordinate containment mechanism to accurately bound rendered elements within the target domain shape (see Figure 1). Otherwise, erroneously extrapolated elements interfere and dominate the visualisation. Explicitly storing coordinate points allows explicit dataset decompression for traditional rasterised rendering without such errors, but provides neither the information needed for implicit ray-marched rendering nor any degree of compression. We therefore consider alternative mechanisms for inside-outside point testing during implicit ray marching. Complex geometries can be specified in a number of ways; three approaches are:

1) **Explicit boundary surfaces:** These are represented by closed polygonal meshes that encode both vertex positions and polygon connectivity [36, 37]. Inside-outside tests can be performed using standard geometric methods such as counting polygon intersections or evaluating winding numbers from rays cast at sampled points [38, 39], which are computationally expensive on-the-fly. Surface points are usually available as labelled boundaries in most simulation suites, making them straightforward to export. The resulting file size scales with the number of surface points and is generally more efficient than storing original coordinates, even when accounting for extra connectivity information. However, such representations remain much larger than those achievable for dataset values through neural compression, thereby limiting overall efficiency.

2) **Discrete occupancy grids:** These partition the domain's minimum bounding box into a regular grid, with each cell storing a binary value indicating whether it lies predominantly inside or outside the domain [40]. Coordinate containment is determined through direct cell queries, which are extremely fast, particularly when using hierarchical structures [41, 42]. Occupancy grids are, however, most often generated by sampling existing

boundary meshes, obtained as above, making them much slower to compute ahead of runtime. Because voxels do not encode surface orientation, boundaries are approximated through thresholding, which introduces significant aliasing along oblique or curved surfaces. Accurately capturing detailed features therefore requires high resolutions, which can easily increase file sizes beyond that of the original data.

3) **Signed distance functions:** These assign a scalar to every point in space, equal to the shortest Euclidean distance to the dataset's domain boundary [37]. By convention, values are negative inside the domain and positive outside, hence the zero level set implicitly defines the boundary [43]. Containment checks are performed by testing whether a point's signed distance is negative. Any shape with a well-defined boundary can, in principle, be represented by a signed distance function. This function may sometimes be expressed analytically for primitive shapes; for complex or non-smooth geometries with sharp edges, however, a single analytical expression is generally unobtainable. In such cases, the domain can be sampled regularly to generate a discrete signed distance grid [44], which can require more storage even than discrete occupancy grids.

These representations trade geometric accuracy against storage and runtime cost, with decimated surface meshes (i.e., reduced-resolution triangle meshes) typically offering a practical compromise. A crucial observation, however, is that signed distances provide a continuous volumetric description of geometry, which, like volumetric data, can be efficiently approximated by neural network models, giving rise to neural signed distance fields [45–47].

DeepSDF [45] demonstrates neural signed distance fields using a latent-code–conditioned feed-forward network with ReLU activations. The model learns continuous mappings from coordinates to signed distances using discrete samples drawn from families of related shapes, with a learned latent vector capturing instance-specific variation. This allows a single network to represent, and smoothly interpolate between, entire classes of geometries (e.g., chairs or cars). Davies et al. [46] simplify the formulation by removing the latent code and overfitting a single network to a single shape, yielding a more accurate and stable representation than DeepSDF, adaptively decimated meshes, or uniform SDF grids of comparable size. They compare sampling strategies for training and evaluate techniques for mitigating MLP spectral bias [48], showing that DeepSDF's original ReLU architecture outperforms positional encodings [29] and SIRENs [30] under equal network capacity. For rendering, they observe that neural SDFs behave similarly to analytical SDFs and support efficient sphere tracing [49], which we exploit.

Although both [45] and [46] highlight the compressive properties of their respective models, neither studies how varying network capacity enables continuous control over compression levels, as later shown by Lu et al. [3]. Motivated by this gap, we adopt a flexible ReLU-based architecture, informed by [3, 46], in which each domain-shape SDF is encoded by a single network whose capacity can be tuned to meet a target compression ratio. This allows us to balance data and domain shape compression within a unified framework for visualisation by implicit ray-march rendering.

III. Methodology

Our method addresses the specific challenge of compressing large volumetric datasets for visualisation through implicit rendering, rather than for numerical analysis or simulation checkpointing [50]. The approach uses two compressed neural representations: one encoding the data and the other encoding the shape of the domain via a signed distance field. By compressing both data and domain with coordinate-based neural implicits, we achieve significant advantages for ray-based rendering techniques that require frequent access at off-grid points and at arbitrary resolution. Together, these components form a compact and memory-efficient representation ideally suited for GPU rendering [51].

Section III.A describes the neural approach to compressing volumetric data values, adapted from [3]. Section III.B then details the neural approach to compressing domain geometry, inspired by [46]. In Section III.C, we present the hybrid ray-marching and sphere-tracing rendering framework. Section III.D introduces Lipschitz-bounded adaptive ray marching and the level-of-detail cascade optimisations used to reduce the total number of network queries per image. Finally, Section III.E describes the implementation of our CUDA-based proof-of-concept isosurface renderer.

A. Compressing Volumetric Data

Our approach to compressing volumetric data closely follows Lu et al. [3] and employs a deep fully connected neural network with sinusoidal activation functions (a SIREN) to map input coordinates to their corresponding scalar values.

We express this as a function $f_\Theta : \vec{x} \to y$, parameterised by network weights Θ, where coordinates $\vec{x} \in \mathbb{R}^3$ are mapped continuously to scalar outputs $y \in \mathbb{R}$. As in Lu et al., we incorporate residual blocks — each comprising two layers with identity skip connections — to improve gradient flow and promote stable convergence by mitigating vanishing and exploding gradients. This combination of sinusoidal activations and residual connections has been shown to most effectively capture complex volumetric structure under tight memory constraints [3]. We also evaluate positional encodings and observe modest improvements over raw input coordinates; accordingly, we apply the first ten log-linear Fourier features ($L = 10$). The resulting architecture is illustrated in Figure 2.

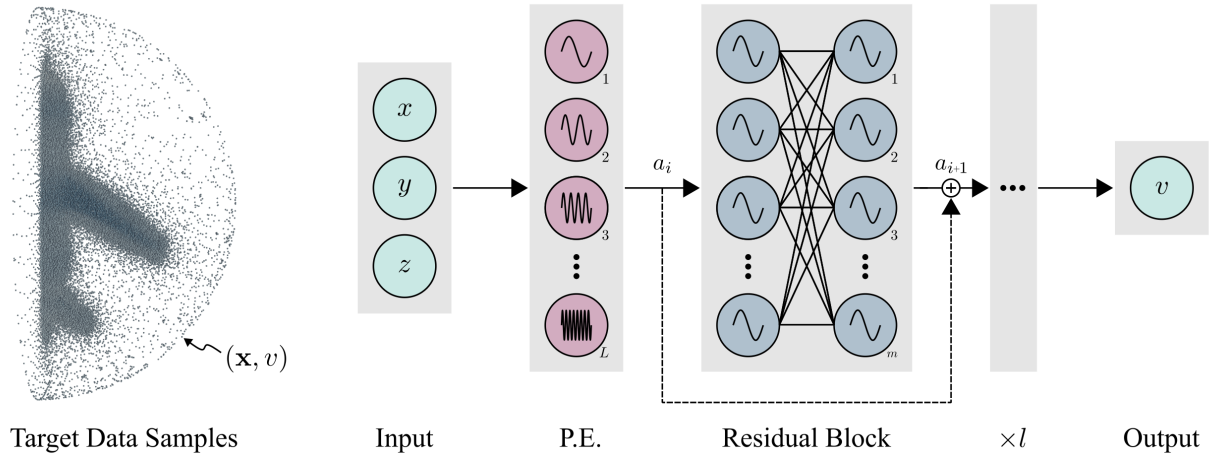

Fig. 2 The Dataset Network consist of l residual blocks, each of width m. All layers are fully-connected with sine activations, except for a linear output layer. Inputs are coordinates $x \in \mathbb{R}^3$ and outputs are corresponding scalar values $y \in \mathbb{R}$. An optional positional encoding layer uses 10 log-linear Fourier features.

The specific network dimensions – and consequently, the total number of parameters – are determined entirely by hyperparameters l and m, which denote the number of residual blocks and the width of each block, respectively. To achieve a compression ratio γ_Θ, we fix l and compute m based on the dataset size, which we define as the number of data points multiplied by the output dimensionality. Within this framework, compression reduces to gradient-descent-based optimisation of Θ over the target coordinate-value pairs using a least-squares loss, while decompression is performed by evaluating f_Θ at arbitrary coordinates. Notably, we forgo Lu et al.'s [3] post-training parameter quantisation scheme, which quadruples compression at a marginal accuracy penalty, and their direct learning of gradients, which improves isosurface accuracy at the cost of runtime performance and ease of implementation for unstructured data.

The *Dataset Network* was implemented in Python using TensorFlow [52] and the Keras API [53], supplemented by custom helper modules and the open-source Trimesh [54] and PyVista [55] libraries. The default network configuration uses eight residual blocks and adopts the SIREN weight initialisation scheme introduced by Sitzmann et al. [30]:

$$W_{i,j} \sim U\left(-\sqrt{\frac{6}{m_i}}, \sqrt{\frac{6}{m_i}}\right), \tag{1}$$

where m_i denotes the size of the input activation to each hidden layer. Parameters are optimised with Adam [56] using an initial learning rate of 0.001 and a variable batch size, trained for 40 epochs without early stopping. Learning-rate decay is applied with a two-epoch half-life. All coordinates are translated and uniformly scaled so that the domain fits within a unit sphere centred at the origin, and all scalar outputs are normalised to the range $[-1, 1]$, in order to support both training stability and correct scaling within downstream rendering.

B. Compressing Domain Shape

Our approach to compressing domain shape builds on the method of Davies et al. [46], and uses a deep fully-connected neural network with ReLU activations [57] to map input coordinates to their corresponding signed distance values of a single domain shape. We express this as a function $g_\Phi : x \to w$ parameterised by network weights Φ, where coordinates

$\vec{x} \in \mathbb{R}^3$ are continuously mapped to signed distances $z \in \mathbb{R}$. By convention, positive distances indicate points outside the domain boundary, while negative values denote points inside, satisfying our occupancy-checking criteria. Consistent with prior work [46], we find that shallow ReLU networks outperform even deep SIRENs for shape representation under a constrained memory budget. In particular, initial tests showed that both SIRENs and Fourier-feature networks struggle to capture smooth surfaces with sharp edges without introducing ripples (i.e., the Gibbs phenomenon) and fail to model the monotonic increase of SDF values for external points. The resulting architecture is illustrated in Figure 3.

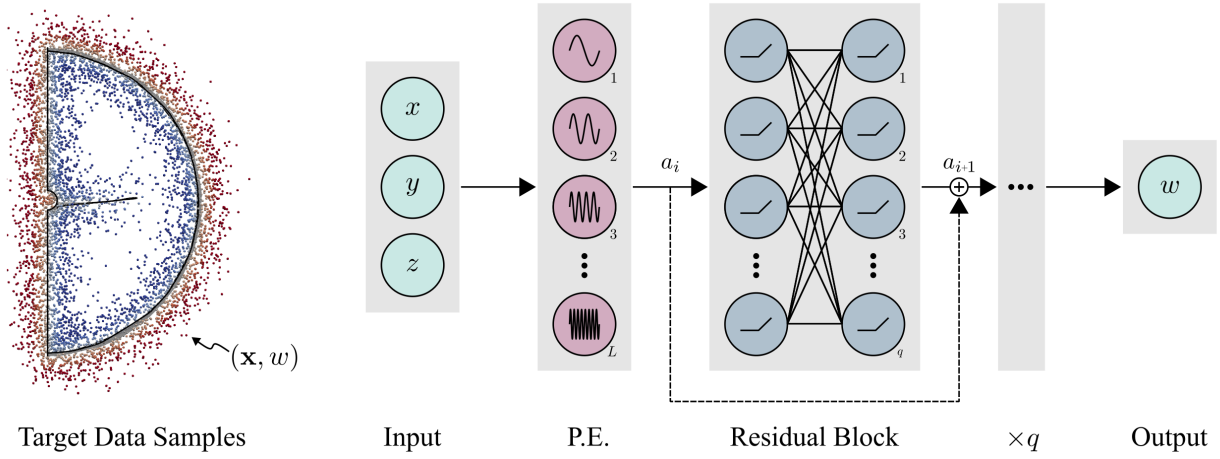

Fig. 3 The Domain Network consist of p residual blocks, each of width q. All layers are fully-connected with ReLU activations, except for a linear output layer. Inputs are coordinates $x \in \mathbb{R}^3$ and outputs are corresponding signed distances $w \in \mathbb{R}$. A positional encoding layer (tested but unused) uses log-linear Fourier features.

We extend Davies et al. [46] to incorporate the same quasi-continuous compression capability used in our Dataset Network. The *Domain Network* architecture is similarly defined by two hyperparameters, p and q, which specify the number of hidden layers and the width of each layer, respectively. To achieve a target compression ratio γ_Φ, we fix p and determine q based on the data volume, calculated as the product of the number of target data points and the input dimensionality $d_{\text{in}} = 3$. The Domain Network is implemented in TensorFlow using Keras, with all signed distances stored in single-precision floating point format. Unless otherwise noted, the default architecture consists of only two layers, which we found to be optimal. Networks are trained using the Adam optimiser with an initial learning rate of 0.001, a batch size of 256, for 40 epochs without early stopping. Learning rate annealing is applied with a two-epoch half-life, and training minimises a least-squares loss. As with the Dataset Network, input coordinates are normalised so that the boundary surface fits exactly within a unit sphere, and signed distances are scaled by the sphere's radius.

Simulation data samples generally lack signed distance values and do not exist outside the domain, which is essential for learning the positive range of SDFs. Computing signed distances on-the-fly is computationally inefficient, so we generate a temporary training dataset prior to training. This process begins by extracting the domain surface as a polygonal mesh using a VTK surface extraction filter [58], followed by triangulation if necessary. We then sample a cloud of 10^6 points, which we find to be sufficient, both inside and outside the mesh. [46] use an *importance sampling* protocol with a costly "sample and replace" rejection strategy to reduce sampling bias on meshes from the Thingi10k dataset [59], but we find that simulation meshes are isotropic enough to rely on Gaussian offsets sampled from surface triangles. Signed distances are then computed using the highly efficient Python Open3D library [60].

C. Implicit Ray-Marched Rendering

A key contribution of this paper is an implicit rendering framework that generates images directly from neural-compressed representations of both data values and domain shape, removing the need for separate explicit domain representations that can limit compression efficiency. We introduce a hybrid ray-marching and sphere-tracing approach, designed to exploit our SDF-based domain representation for more efficient sampling along rays while producing accurate renderings of target data. We focus on isosurface rendering because it is a challenging yet common 3D task, and it serves as a proof of concept for simpler rendering tasks, such as cut planes. The rendering process is summarised in Figure 4.

In its most basic form, the rendering scene is defined by a virtual camera, a virtual light source, a bounding sphere object enclosing all points within the domain, and implicit coordinate-based representations of both data values and domain shape. These representations can be either compressed or uncompressed, and can include non-neural formulations (e.g., analytical functions). In our demonstration, both data and domain are neural representations, compressed roughly — but not necessarily identically — to the same target compression ratio.

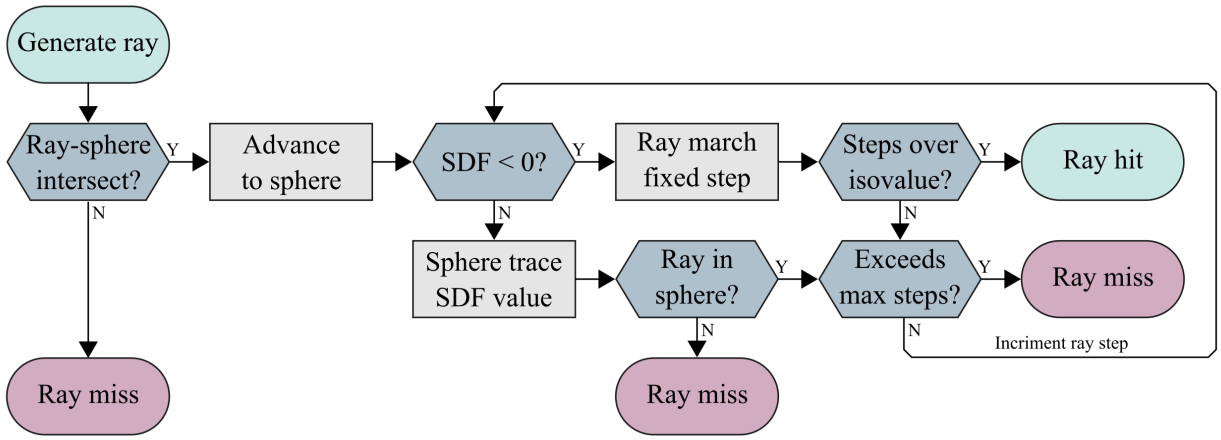

Fig. 4 Our hybrid sphere-tracing and ray-marching approach for implicit isosurface rendering on neural representations of data and domain. Rays either miss or intersect the bounding sphere. If a ray intersects, then it will eventually either step into the target domain (SDF<0) or exit through the rear face of the bounding sphere (ray miss). Within the domain (SDF>0) rays may either step over the target isovalue (rah hit), reach the maximum step limit (ray miss), or step back outside the domain. Only ray hits within the domain are shown on screen.

The baseline isosurface-rendering pipeline proceeds as follows. A camera-originated ray is traced, for each on-screen pixel, through the (w, h)-resolution image plane placed at unit distance and perpendicular to the viewing axis. An initial intersection test is performed against a bounding sphere whose centre and radius minimally enclose the domain surface. Rays that miss are discarded; those that hit advance to the near-side intersection. Beyond reducing unnecessary marching, this culling step ensures that SDF samples are drawn from the smallest controlled region and that learning is concentrated around the surface geometry rather than the far field. The surviving rays then advance via our iterative hybrid ray-marching and sphere-tracing scheme. At each step, the neural SDF is evaluated at the ray's current position to obtain both the nearest-surface distance and occupancy. A positive signed distance indicates that the ray is outside the domain, allowing it to advance safely by that distance under the sphere-tracing rule. The ray remains active provided it stays within the bounding sphere and does not exceed the user-defined maximum number of steps; otherwise it is classified as a miss and culled. A negative signed distance indicates that the ray is inside the domain, in which case it advances by a fixed step size δ under the ray-marching rule while also querying the neural compressed value at that location. The step size is chosen a priori as the 0.1 percentile of cell-edge lengths, which filters out extreme outliers, behaves consistently across differently scaled domains, and is sufficiently small to avoid overstepping thin interior structures. An isosurface hit is detected when the sign of the residual between the compressed data value and the target isovalue changes across consecutive steps. Crucially, the framework forbids a hit from occurring while the ray is outside the target domain. Rays that do not register a hit continue marching; those that do are terminated, and basic lighting is computed using surface normals estimated from finite differences along each coordinate axis. The process continues, with step counts incremented each iteration, until every pixel has either registered a hit or satisfied a miss condition.

Our isosurface approach generalises naturally to a wide range of rendering tasks, including surface, cut-plane, and even full volumetric rendering. The only requirement is that any data extrapolated beyond the SDF domain boundary is treated as unreliable and therefore ignored by the renderer. By exploiting the bounding sphere, it is also straightforward to mix neural and analytical SDFs within the same coordinate-containment checks, enabling a variety of composite effects. In turn, this makes it possible to render multiple datasets simultaneously within a single unified framework.

D. Improving Rendering Performance

Rendered images may contain hundreds of thousands of pixels, depending on the desired resolution, and each pixel can require hundreds of ray-marching steps that repeatedly query the neural representations of data or domain. These evaluations accumulate along a ray because each step depends on information obtained at the previous one, limiting opportunities for intra-ray parallelism. While bounding-sphere culling removes many guaranteed misses from the outset, repeated successive neural network inference remains the dominant performance bottleneck. Because sphere tracing is theoretically optimal with respect to its step-selection rule, the vast majority of sequential queries arise from marched rays advancing with small, fixed step size inside the domain. We deliberately avoid caching intermediate network outputs because it introduces substantial memory overhead and offers little benefit given the lack of overlap between neighbouring ray trajectories. Instead, we introduce two techniques to reduce the number of per-pixel queries specifically within the ray marching regime: (i) an adaptive ray-marching strategy that replaces the traditional fixed-step approach, and (ii) a depth-informed level-of-detail (LoD) cascade analogous to progressive rendering. Both techniques operate under a global Lipschitz bound, which guarantees that reductions in query count do not come at the cost of ray overstepping, as would arise from simply increasing the step size globally.

Of key importance is the role of the Lipschitz bound in regulating sampling, as it strictly limits the maximum change in function outputs (e.g., data values) relative to absolute changes in function inputs (e.g., ray coordinates). By definition, a function $f : \mathbb{R}^n \to \mathbb{R}$ satisfies the Lipschitz condition globally if it is piecewise continuous and there exists a constant $L \geq 0$ such that:

$$|f(\vec{x_1}) - f(\vec{x_2})| \leq L \, \|\vec{x_1} - \vec{x_2}\| \qquad \forall \vec{x_1}, \vec{x_2} \in \mathbb{R}^n. \tag{2}$$

We enforce this condition across all samples from the target dataset, allowing us to determine L via finite differences. The resulting Lipschitz constant, which can be interpreted as a first-order, direction-independent maximum gradient, is combined with the minimum cell size δ_{min} to compute per-step, per-ray adaptive step sizes. Each step is chosen to be as large as possible without allowing sampled function to change by more than the target isovalue residual. We take the maximum of this calculated step size and δ_{min} to prevent excessively small steps from stalling ray progression when residuals are small near the surface. This ensures that each step is safe and efficient: preventing overstepping while maintaining steady forward progress. Formally, the adaptive step sizes s are given by:

$$s = \max\left(\delta_{\text{min}}, \frac{|f(\vec{x}_{\text{current}}) - f(\vec{x}_{\text{target}})|}{L}\right), \tag{3}$$

where $f(\vec{x}_{\text{current}})$ and $f(\vec{x}_{\text{target}})$ are the current ray data value and target isovalue, respectively. This formulation maximizes step size without skipping any target isovalue, ensuring that rays crossing an isosurface are sampled within at least two steps. By taking the maximum of the Lipschitz-informed step and the minimum cell size, we guarantee an equal or reduced number of steps per ray compared to the baseline. The reduction depends on the specific data field but is always bounded by the original step count and incurs minimal computational overhead for implementation.

The number of per-image network queries can also be reduced by reusing ray-depth information obtained from an initial low-resolution pass, where fewer pixels are evaluated, to inform subsequent full-resolution rendering. Our level-of-detail (LoD) cascade implements this approach by progressively refining intermediate results, using depth values from each pass to initialise rays in the next, so that high-resolution results require fewest per-pixel steps. The process begins without prior depth information and renders an image at $2^{(1-\text{LoD})}$ of the native resolution. Each pass produces both an RGBA image and a single-channel depth buffer. The depth buffer records, for each ray, the furthest depth reached along the ray direction during the previous pass. Before reuse, these depths are retracted by a small amount to ensure that rays generated in the subsequent pass are not initialised deeper than the isosurface they may have just passed through to register a hit:

$$\hat{d}_r = d_r - \delta_{\text{min}} - \left(\frac{d_r \, p_r \, L \, \delta_{\text{min}}}{2 \, \|\nabla f(\vec{x}_r)\|}\right). \tag{4}$$

Here, p_r denotes the pixel footprint at depth d_r and $\nabla f(\vec{x}_r)$ is the backwards-difference rate of change of ray value at the final marched position along the ray. In this formulation, the Lipschitz bound limits the maximum variation of the field

within the pixel footprint perpendicular to the ray direction. The local gradient then relates this bound to an equivalent variation along the ray, which determines the backward adjustment of the initialisation depth. As illustrated in Figure 5, this adjustment is essential to prevent newly spawned rays from being initialised beyond the target isosurface.

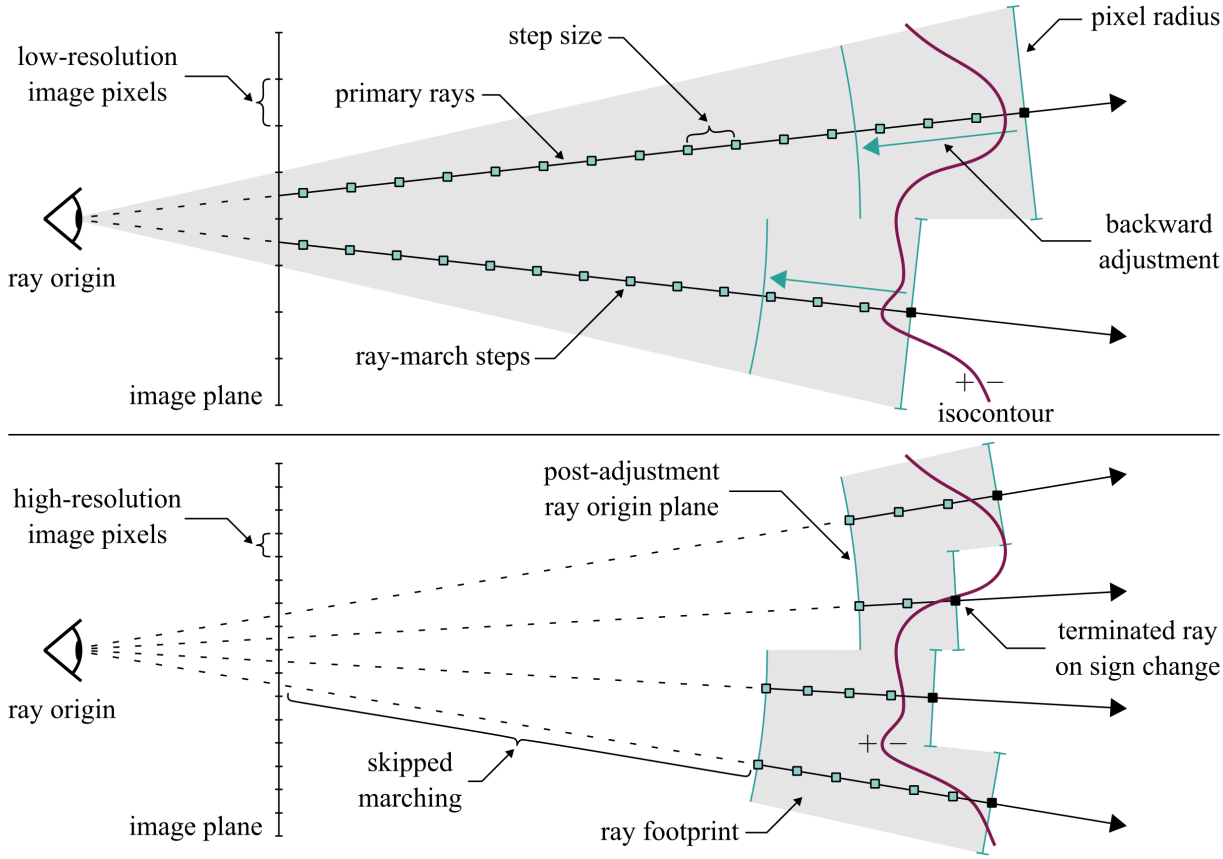

Fig. 5 A side-on illustration of our depth-informed level-of-detail cascade, showing backward adjustment from an initial low-resolution pass (top) to subsequent higher-resolution rendering (bottom). Rays may step beyond the isosurface before registering a hit and are adjusted backwards before their depth values are recycled to avoid overshooting. Initialising new rays from this adjusted position significantly reduces ray-marching iterations compared to naive marching from the original image plane, even as the pixel count increases.

Both the image and depth buffers are then upscaled using nearest-neighbour extrapolation, for display and as input to the subsequent render pass, respectively. The depth buffer additionally undergoes minimum pooling with a 3×3 cross-shaped kernel to further prevent overstepping at tangential seams near isosurface boundaries. The resulting depth values are used to initialise rays at the next level of detail much closer to their final depths, rather than from the image plane, and the image is progressively refined until full resolution is reached. In our experiments, we find that a four-stage LoD cascade with an upscaling factor of two provides the best trade-off between the increased number of rendering passes, increased pixel count per pass, and the progressively decreasing number of steps per pixel.

E. Proof-of-Concept GPU Rendering

Implicit rendering from compressed representations is computationally intensive yet highly parallelisable across pixels and has low memory overhead, making it well suited to hardware acceleration [51]. Similarly, neural network evaluation at independent sample locations benefits from GPUs' high core counts, enabling efficient execution even when threads perform separate inferences with minimal data reuse. We exploit this synergy by implementing the isosurface rendering framework entirely on the GPU using CUDA in C++, with OpenGL handling user interaction and visualisation. The source code and additional details for this proof-of-concept demo can be made available upon request.

Our parallelised approach partitions each image into multiple thread blocks, with each pixel mapped to a dedicated CUDA thread within its thread block. We omit multisampling anti-aliasing to maximise frame rates. Queries to the neural representations are executed using single-threaded matrix-vector operations with activation functions, with network parameters stored entirely in GPU memory. Depth and colour buffers reside in global GPU memory and are updated across successive levels of detail to minimise redundant CPU–GPU communication. Each thread block is allocated shared memory that functions as a ping-pong buffer, allowing threads to store intermediate activation vectors efficiently without memory leaks. Finally, CUDA–OpenGL interoperability enables seamless data transfer, supporting real-time visualisation and interactive control via mouse and keyboard input.

IV. Results and Discussion

This section evaluates the proposed approach through three analyses. In Section IV.A, neural representations of both data and domain are visually compared against alternative value and shape representations from prior work, under equivalent compression levels. In Section IV.B, the effect of the proposed performance optimisations is measured by counting network queries per fully rendered image using a Python-based renderer. In Section IV.C, the CUDA implementation is benchmarked to assess its suitability for interactive rendering. Details of the datasets used are provided in Table 1.

Table 1 Summary of the datasets used for performance evaluation, including domain type, combined coordinate-value grid resolution, and uncompressed file size in Megabytes (MB).

Dataset Name	Domain Type	Grid Resolution	File Size (MB)
Aorta	Unstructured	$(1134045, 4)$	18.1
Buoyancy	Structured	$(256, 256, 256, 4)$	268
DNS-6	Structured	$(266, 140, 60, 4)$	35.8
Rotor-67	Unstructured	$(391498, 4)$	6.3

We do not use standard publicly available benchmark datasets commonly used in compression studies, because they almost exclusively assume structured grids defined over regular, axis-aligned, rectangular domains. In such cases, the domain boundary is trivial to represent, and there is no justification for compressing geometry with a neural signed distance field instead of using a simple bounding box. Many aerospace and turbomachinery simulations, however, are defined on unstructured, anisotropically sampled meshes with complex shaped boundaries. These domains are substantially more difficult to constrain during rendering and directly motivate the use of the proposed technique.

A. Comparing Compressed Representations of Data and Domain

We evaluate the proposed approach by jointly assessing neural compression of volumetric data and neural compression of the spatial domain, as well as the effectiveness of constraining compressed data to a compressed domain during isosurface visualisation. The evaluation combines ray-marching and ray–triangle rendering to enable direct comparisons between implicit and explicit representations. Domain representations are compared against three alternatives: the ground-truth triangle surface mesh, an implicit bounding-sphere domain, and a decimated version of the original surface mesh, which serves as a geometry-based baseline. Data representations are compared against two alternatives: a surface mesh of the ground-truth isosurface extracted from uncompressed data, and surface meshes of isosurfaces extracted from data compressed using SZ3 [61] and MGARD [18], which serve as data-based baselines.

Because signed-distance values required for sphere tracing are unavailable when rendering explicit surface meshes, and because sampling onto a grid would compromise accuracy, we adapt our approach to standard ray marching for these cases. Rays that pass bounding-sphere culling are marched directly to the nearest ray–triangle intersection with the mesh in a single step. Simplified surface meshes are generated by progressively decimating the original surface mesh—reducing both triangle and vertex counts — until their on-disk storage size matches that of the compressed SDF representation of the same geometry, ensuring a fair comparison based solely on compression ratio. The spherical domain representation, as the simplest possible bounding structure, serves to demonstrate that insufficient domain constraint typically leads to erroneous isosurface extrapolation outside the true geometry.

Figure 6 compares domain shape representations for the DNS-6, Rotor-67, and Aorta datasets under medium and high compression ratios ($\gamma \in \{50, 200\}$). All images are generated using our custom rendering engine; for the neural SDF cases, rendering employs only the sphere-tracing regime.

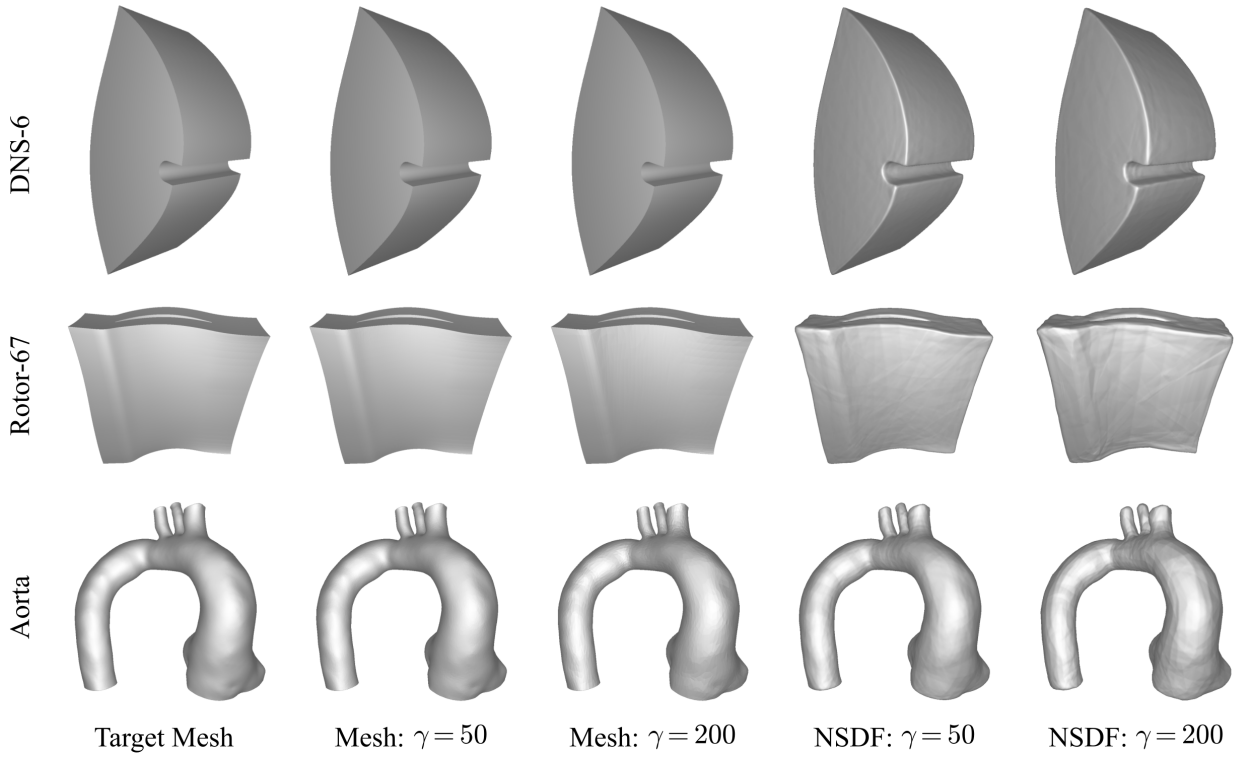

Fig. 6 Comparison of domain shape representations for the DNS-6, Rotor-67, and Aorta datasets. The ground-truth mesh (left) serves as the reference for the decimated mesh ("Mesh") and neural SDF ("NSDF") renderings at compression ratios $\gamma \in \{\mathbf{50}, \mathbf{200}\}$.

Both the decimated mesh and neural SDF representations preserve the original spatial domain at medium and high compression ratios. The close visual agreement with the ground-truth meshes shows that the neural SDF captures complex geometry with minimal distortion. On the larger DNS-6 and Aorta datasets, it maintains both global shape and local surface quality, with only minor smoothing at sharp edges. On the smaller Rotor-67 dataset, which has a higher surface-to-volume ratio and includes thin hollow structures, degradation becomes noticeable at the highest compression level, appearing as increased surface roughness and less defined corners and inclusions. Even at this level, the representation remains sufficient for bounding the domain. The decimated mesh approach performs slightly better, particularly on the smaller Rotor-67 dataset, showing near-imperceptible degradation in global structure or local surface quality relative to the target mesh across the tested compression range. This contrasts with the results reported in [46], likely because our compression equivalence is based on the original file size of all coordinates, including internal points, rather than just the surface mesh with surface vertices and connectivity. Although the decimated mesh preserves geometry exceptionally well, it introduces substantial complexity when integrated into the implicit rendering pipeline. In comparison, the neural SDF offers seamless integration, preserves domain boundaries with sufficient fidelity, and supports a fully implicit workflow, making it the more practical choice for compressed rendering.

Figure 7 additionally compares neural representations of the structured Buoyancy and unstructured Aorta datasets versus SZ3 and MGARD at a medium compression ratio (γ = 50). This comparison highlights the need of neural compression and, by extension, a neural compressed domain representation.

All three compression methods yield accurate isosurface renderings on the structured Buoyancy dataset, consistent with prior work. On the unstructured Aorta data, however, SZ3 and MGARD fail to preserve the isosurface even at modest compression ratios, while the neural approach again succeeds with minimal degradation. This is especially relevant for compressing aerospace and turbomachinery datasets, which are often unstructured, non-uniform, and

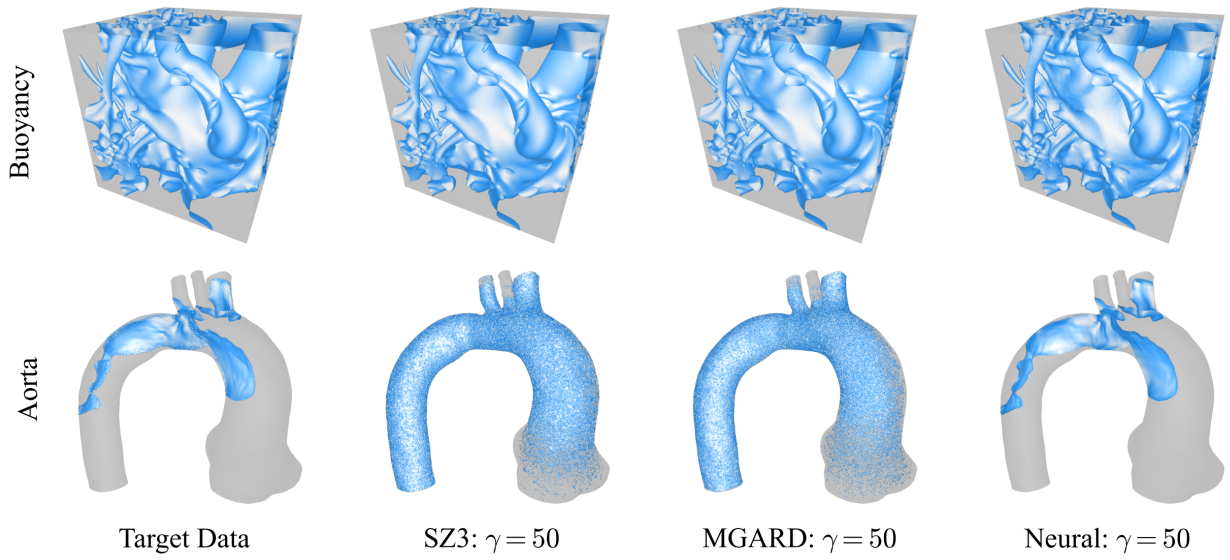

Fig. 7 Comparison of domain value compression for the Buoyancy and Aorta datasets. The ground-truth isosurface (left, blue) serves as reference for SZ3, MGARD, and neural compressed renderings at $\gamma = 50$.

defined on complex shaped domains. Previous compression studies largely ignore such real-world data and thus the need to compress both values and domain jointly. Our results show that the proposed dual-neural-network approach effectively addresses this challenge.

B. Evaluating Ray-March Rendering Performance Optimisations

We quantify the performance gains of the depth-informed LoD cascade and Lipschitz adaptive ray-marching strategies, both individually and combined. The evaluation targets the primary performance bottleneck: the total number of ray-tracing steps per full-resolution image. For this test, we render a 1080×1080 pixel image using (i) the baseline renderer, (ii) the baseline with adaptive ray marching, (iii) the baseline with a four-stage LoD cascade, and (iv) both optimisations together. Figure 8 reports the total number of queries required to render an isosurface on the Aorta and Buoyancy datasets. For schemes (iii) and (iv) that incorporate the level-of-detail cascade, each final image is preceded by three smaller images, effectively increasing the total number of rendered pixels by a factor of $4/3$; the breakdown of ray steps is represented using stacked bars with upwards increasing resolutions. Visual comparisons are omitted because all methods prevent ray overshooting and thus produce identical outcomes on identical compressed data.

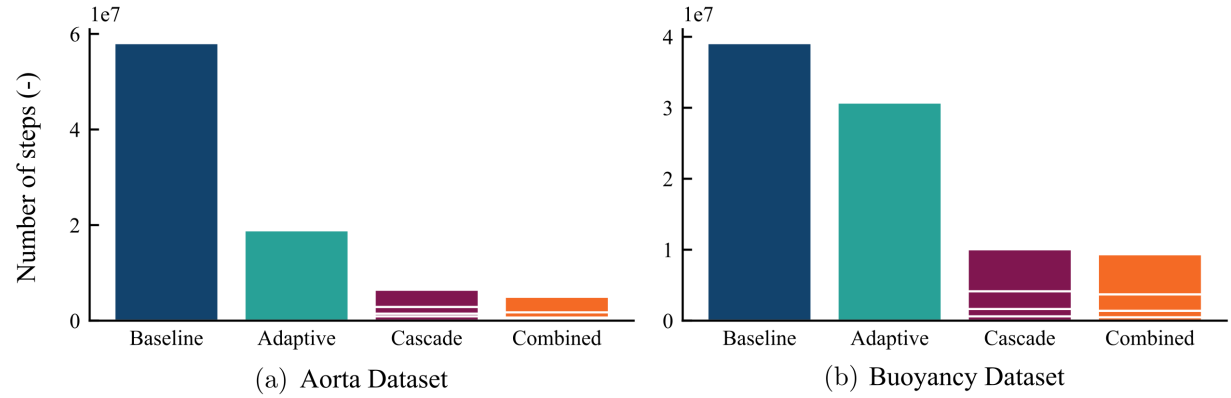

Fig. 8 Comparison of per-image ray-step counts (combining both ray-marching and sphere-tracing) for individual and combined rendering optimisations, versus the baseline, on the Aorta and Buoyancy datasets.

The adaptive-step marching and level-of-detail cascade optimisations reduce the total number of network queries per full-resolution image on the Aorta dataset by 67.4% and 88.8%, respectively, relative to the baseline, with the combined approach achieving a reduction of 91.3%. Similar trends are observed on the Buoyancy dataset, with reductions of 21.4%, 74.3%, and 76.1% for each successive optimisation. Since the proposed rendering framework is primarily dominated by neural network inference, and each ray-marching step issues exactly one network query, these reductions translate directly into fewer floating-point operations. Across both datasets, the LoD cascade consistently initialises rays close to their final termination points rather than at the image plane, substantially reducing the number of required steps for both hits and misses. Adaptive marching also lowers step counts, but its effectiveness depends strongly on the data's Lipschitz bound, which varies between datasets. This sensitivity arises because the Lipschitz bound directly determines per-step traversal distance. In contrast, the LoD cascade is largely insensitive to such variation, as the Lipschitz bound only affects the backward adjustment between successive levels of detail, which is applied once per pass.

Figure 9 shows pixel-level performance gains of the combined optimisations relative to the baseline rendering approach, using heatmaps of ray-marched steps, for the Buoyancy, DNS-6, and Aorta datasets. Darker pixels indicate a higher number of combined ray-marched and sphere-traced steps and thus greater computational cost per pixel.

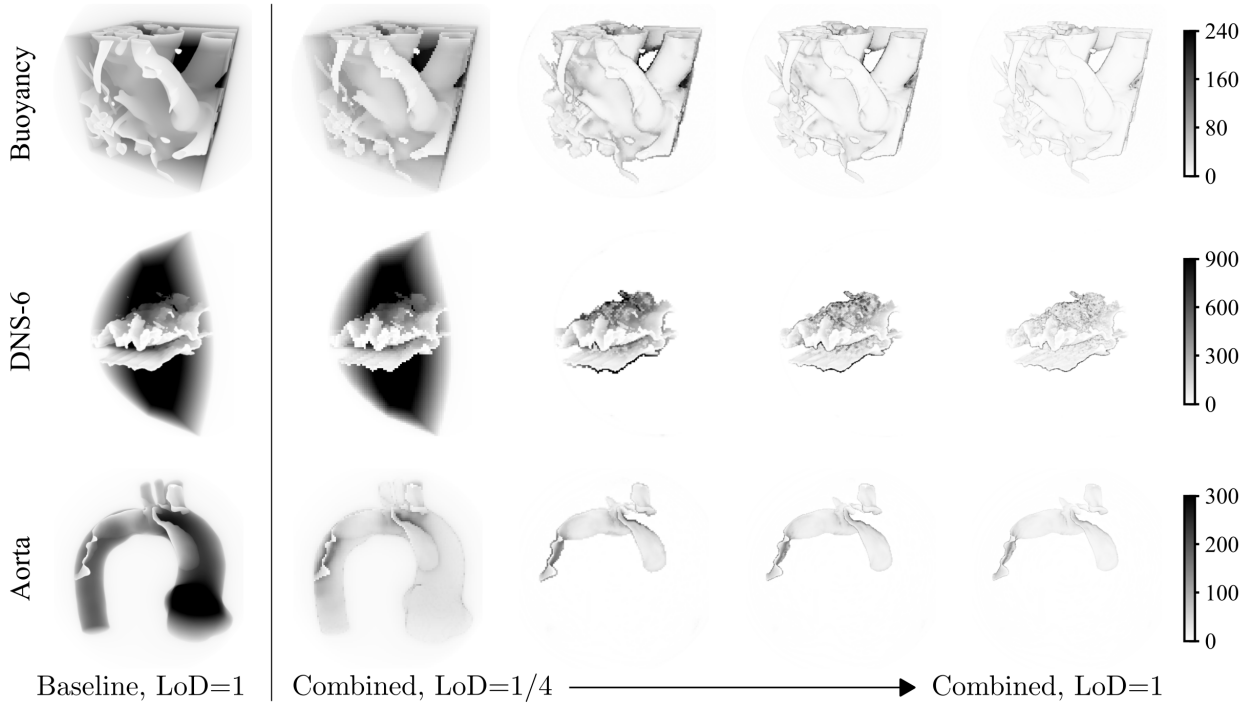

Fig. 9 Heatmaps of ray-marched steps comparing the baseline and combined rendering approaches, for the Buoyancy, DNS-6, and Aorta datasets. Step count includes both ray-marched and sphere-traced steps, with the former contributing the vast majority to rendering cost via substantially more network queries.

The source of improved rendering efficiency becomes clearer when comparing the combined approach against the baseline renderer at increasing levels of detail. In the first pass, the number of steps per pixel is similar in both cases (baseline LoD = 1 versus combined LoD = 1/4), since no prior depth information is available; any reduction in step count at this stage comes solely from adaptive marching, which is most effective on the low-Lipschitz-bound Aorta dataset. At subsequent levels of detail, substantial gains appear in pixels corresponding to rays that traverse the entire depth of the domain without intersecting an isosurface. In the baseline heatmaps, these rays form a dark silhouette of the domain and dominate the computational cost, because they march the maximum possible distance before termination. With the LoD cascade, such rays are initialised closer to the far side of the domain and exit after only a few steps, quickly re-registering as misses. Ray hits are similarly initialised closer to their final positions and re-register as hits after fewer steps. More limited improvements occur near tangential isosurface boundaries, where shallow gradients along the ray lead to larger backward adjustments (see Eq. 4) and where minimum pooling during depth upscaling restricts reuse. Even in these small regions, the LoD cascade consistently provides a clear efficiency advantage over the baseline.

It is important to note that although the number of rays per image quadruples at each resolution increase within the LoD cascade, the total number of rendered pixels rises by only $4/3$ compared to any single-pass approach. Across these stages, the average number of network queries per pixel drops from 26.3 to 12.5, then to 7.9, and finally to 4.8 on the buoyancy dataset. Per-pixel performance continues to improve beyond the first depth reuse because higher-resolution pixels have smaller footprints, producing smaller backward adjustments and, in turn, better ray initialisations.

C. Benchmarking Performance in the CUDA Renderer

Our CUDA-based isosurface renderer is benchmarked at 800×800 pixels on a local machine with a 12th Gen Intel Core i7-12700 CPU and an NVIDIA RTX A2000 GPU with 12 GB of VRAM. The domain is represented by a compressed neural SDF trained on a finely triangulated unit cube mesh to ensure well-distributed sample points. This representation, defined by 1,153 trainable parameters, is shown in Fig. 10(a). The data values are generated on a regular 256^3 lattice within the unit cube using the implicit polynomial equation for a tangle-cube*, with the field values at each point forming the volumetric representation. When compressed, this data requires only 703 trainable parameters and is visualised for an arbitrary isovalue in Fig. 10(b). The tangle-cube provides a compact but challenging test case, with varied depth and internal holes that stress the renderer and allow evaluation of performance optimisations. It is also convenient for development, as both data and cube-SDF domain can be described exactly via algebraic functions.

As in Section IV.B, we evaluate four configurations: (i) the baseline renderer, (ii) the baseline with adaptive ray marching, (iii) the baseline with a four-stage LoD cascade, and (iv) all optimisations combined. To simulate user interaction, full-resolution frames are rendered sequentially across 200 camera positions evenly distributed on a Fibonacci lattice around the domain. Figure 10 shows the average render time per frame for each configuration.

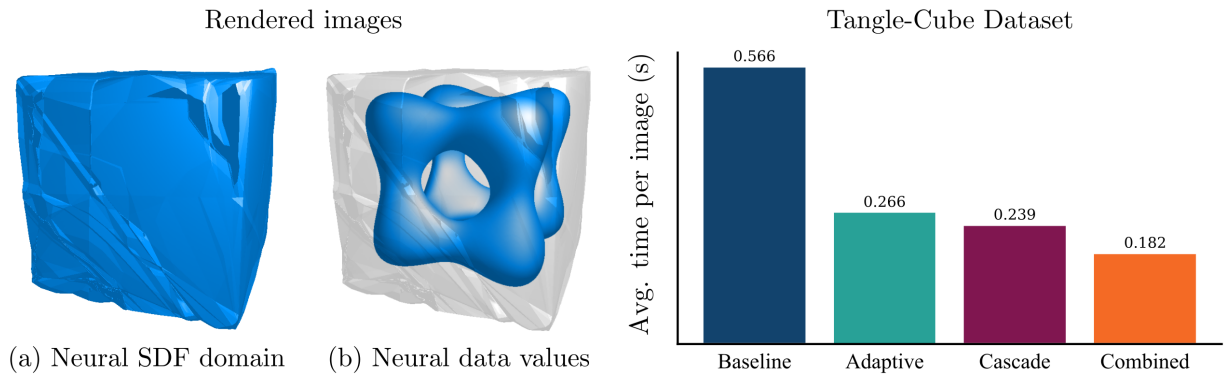

Fig. 10 Average seconds per frame for the baseline renderer, adaptive ray marching, LoD cascade, and their combination. Subfigure (a) shows the compressed domain representation, and (b) shows the domain together with an isosurface on the tangle-cube data.

Step-count trends from the Python renderer are consistent with CUDA benchmarks, confirming scalability under parallelisation. Adaptive marching yields a larger than expected gain, reducing frame time by 52.9% relative to the baseline. This is likely due to the tangle-cube's smooth field, which allows larger step sizes under a lower Lipschitz bound than previously-shown datasets. The LoD cascade reduces frame time by 67.7%, which is less than indicated by step-count trends in Figure 9. This difference is likely because of CUDA's thread-block execution: all threads within a block must finish before the block completes, so slower pixels create idle time that limits actual speedup.

The combined approach achieves 5.47 frames per second, showing the cumulative benefit of combining multiple optimisation strategies. While not interactive at full resolution, we deliberately reset the render engine to the lowest LoD whenever the camera changes (e.g., via panning and zooming). This allows users to navigate smoothly in real time: during navigation, frames are rendered at reduced 1/4 resolution, achieving 51.6 FPS. Once the camera stabilises, the renderer progressively refines the image to higher resolution until full resolution is reached.

*The equation for a tangle-cube is: $f(\vec{x}) = x^4 + y^4 + z^4 - (x^2 + y^2 + z^2)\,, \quad f(\vec{x}) = -0.4.$

V. Conclusions

This research highlights the importance of a domain shape representation for rendering neural compressed volumetric data, to prevent extrapolated artifacts. The results we achieve demonstrate the feasibility of implicit rendering with neural domain shape representations and neural compressed volumetric data at high compression ratios: $\gamma = 200$. The key contributions of our paper are:

1) A fully-implicit, dual-neural-network framework that constrains compressed representations of volumetric data values within the original domain extent. This is achieved with a compressed neural SDF.
2) A hybrid sphere-tracing and ray-marching approach for rendering isosurfaces from neural-compressed data. We show that this technique can be extended to other surface- and volume-rendering applications.
3) A level-of-detail (LoD) cascade and Lipschitz-informed adaptive marching regime that significantly reduce the number of ray steps per image, enabling higher frame rates while preserving image quality.
4) Validation via a proof-of-concept viewer, leveraging shared interoperability between CUDA and OpenGL to achieve interactive rendering at 51 FPS on an NVIDIA RTX A2000 GPU.

Acknowledgments

We would like to thank Owais Khan of the Department of Electrical, Computer, & Biomedical Engineering at Toronto Metropolitan University for providing the Aorta dataset, and Thanassis Frank of the University of Cambridge's Whittle Laboratory for the many helpful technical discussions that contributed to the development of this work.

References

[1] Almgren, A. S., Bell, J. B., Lijewski, M. J., Lukić, Z., and Van Andel, E., "Nyx: A MASSIVELY PARALLEL AMR CODE FOR COMPUTATIONAL COSMOLOGY," *The Astrophysical Journal*, Vol. 765, No. 1, 2013, p. 39. https://doi.org/10.1088/0004-637x/765/1/39.

[2] Dyer, G., and Fry, A., *Matter in Extreme Conditions Upgrade (Conceptual Design Report)*, 2021. https://doi.org/10.2172/1866100.

[3] Lu, Y., Jiang, K., Levine, J. A., and Berger, M., "Compressive Neural Representations of Volumetric Scalar Fields," *Computer Graphics Forum*, Vol. 40, No. 3, 2021, pp. 135–146. https://doi.org/10.1111/cgf.14295.

[4] Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R., "NeRF: representing scenes as neural radiance fields for view synthesis," *Communications of the ACM*, Vol. 65, No. 1, 2021, pp. 99–106. https://doi.org/10.1145/3503250.

[5] Garbin, S. J., Kowalski, M., Johnson, M., Shotton, J., and Valentin, J., "FastNeRF: High-Fidelity Neural Rendering at 200FPS," *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, IEEE, 2021, pp. 14326–14335. https://doi.org/10.1109/iccv48922.2021.01408.

[6] Reiser, C., Peng, S., Liao, Y., and Geiger, A., "KiloNeRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs," *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, IEEE, 2021. https://doi.org/10.1109/iccv48922.2021.01407.

[7] Kerbl, B., Kopanas, G., Leimkuehler, T., and Drettakis, G., "3D Gaussian Splatting for Real-Time Radiance Field Rendering," *ACM Transactions on Graphics*, Vol. 42, No. 4, 2023, pp. 1–14. https://doi.org/10.1145/3592433.

[8] Niemeyer, M., Manhardt, F., Rakotosaona, M.-J., Oechsle, M., Duckworth, D., Gosula, R., Tateno, K., Bates, J., Kaeser, D., and Tombari, F., "RadSplat: Radiance Field-Informed Gaussian Splatting for Robust Real-Time Rendering with 900+ FPS," , 2024. https://doi.org/10.48550/ARXIV.2403.13806.

[9] Moenne-Loccoz, N., Mirzaei, A., Perel, O., de Lutio, R., Martinez Esturo, J., State, G., Fidler, S., Sharp, N., and Gojcic, Z., "3D Gaussian Ray Tracing: Fast Tracing of Particle Scenes," *ACM Transactions on Graphics*, Vol. 43, No. 6, 2024, pp. 1–19. https://doi.org/10.1145/3687934.

[10] Lakshminarasimhan, S., Shah, N., Ethier, S., Ku, S., Chang, C. S., Klasky, S., Latham, R., Ross, R., and Samatova, N. F., "ISABELA for effective in situ compression of scientific data," *Concurrency and Computation: Practice and Experience*, Vol. 25, No. 4, 2012, pp. 524–540. https://doi.org/10.1002/cpe.2887.

[11] Wallace, G., "The JPEG still picture compression standard," *IEEE Transactions on Consumer Electronics*, Vol. 38, No. 1, 1992, pp. xviii–xxxiv. https://doi.org/10.1109/30.125072.

[12] Gall, D. J. L., "The MPEG video compression algorithm," *Signal Process. Image Commun.*, Vol. 4, No. 2, 1992, pp. 129–140. https://doi.org/10.1016/0923-5965(92)90019-C.

[13] Haar, A., "Zur Theorie der orthogonalen Funktionensysteme: Erste Mitteilung," *Mathematische Annalen*, Vol. 69, No. 3, 1910, pp. 331–371. https://doi.org/10.1007/bf01456326.

[14] Daubechies, I., *Ten lectures on wavelets*, SIAM, 1992.

[15] Mulcahy, C., "Image compression using the Haar wavelet transform," *Spelman Science and Mathematics Journal*, Vol. 1, No. 1, 1997, pp. 22–31.

[16] Lindstrom, P., "Fixed-Rate Compressed Floating-Point Arrays," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 20, No. 12, 2014, pp. 2674–2683. https://doi.org/10.1109/tvcg.2014.2346458.

[17] Ballester-Ripoll, R., Lindstrom, P., and Pajarola, R., "TTHRESH: Tensor Compression for Multidimensional Visual Data," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 26, No. 9, 2020, pp. 2891–2903. https://doi.org/10.1109/tvcg.2019.2904063.

[18] Gong, Q., Chen, J., Whitney, B., Liang, X., Reshniak, V., Banerjee, T., Lee, J., Rangarajan, A., Wan, L., Vidal, N., Liu, Q., Gainaru, A., Podhorszki, N., Archibald, R., Ranka, S., and Klasky, S., "MGARD: A multigrid framework for high-performance, error-controlled data compression and refactoring," *SoftwareX*, Vol. 24, 2023, p. 101590. https://doi.org/10.1016/j.softx.2023.101590.

[19] Li, S., Lindstrom, P., and Clyne, J., "Lossy Scientific Data Compression With SPERR," *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2023, pp. 1007–1017. https://doi.org/10.1109/IPDPS54959.2023.00104.

[20] Lindstrom, P., and Isenburg, M., "Fast and Efficient Compression of Floating-Point Data," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 12, No. 5, 2006, pp. 1245–1250. https://doi.org/10.1109/tvcg.2006.143.

[21] Di, S., and Cappello, F., "Fast Error-Bounded Lossy HPC Data Compression with SZ," *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE, 2016, pp. 730–739. https://doi.org/10.1109/ipdps.2016.11.

[22] Liang, X., Di, S., Tao, D., Li, S., Li, S., Guo, H., Chen, Z., and Cappello, F., "Error-Controlled Lossy Compression Optimized for High Compression Ratios of Scientific Datasets," *2018 IEEE International Conference on Big Data (Big Data)*, IEEE, 2018, pp. 438–447. https://doi.org/10.1109/bigdata.2018.8622520.

[23] Hornik, K., Stinchcombe, M., and White, H., "Multilayer feedforward networks are universal approximators," *Neural Networks*, Vol. 2, No. 5, 1989, pp. 359–366. https://doi.org/10.1016/0893-6080(89)90020-8.

[24] Broomhead, D. S., and Lowe, D., "Multivariable Functional Interpolation and Adaptive Networks," *Complex Syst.*, Vol. 2, 1988. URL https://api.semanticscholar.org/CorpusID:3686496.

[25] Cybenko, G., "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, Vol. 2, No. 4, 1989, pp. 303–314. https://doi.org/10.1007/bf02551274.

[26] Funahashi, K.-I., "On the approximate realization of continuous mappings by neural networks," *Neural Networks*, Vol. 2, No. 3, 1989, pp. 183–192. https://doi.org/10.1016/0893-6080(89)90003-8.

[27] Poggio, T., and Girosi, F., "Networks for approximation and learning," *Proceedings of the IEEE*, Vol. 78, No. 9, 1990, pp. 1481–1497. https://doi.org/10.1109/5.58326.

[28] Sitzmann, V., Zollhöfer, M., and Wetzstein, G., "Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations," , 2019. https://doi.org/10.48550/ARXIV.1906.01618.

[29] Tancik, M., Srinivasan, P. P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J. T., and Ng, R., "Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains," , 2020. https://doi.org/10.48550/ARXIV.2006.10739.

[30] Sitzmann, V., Martel, J. N. P., Bergman, A. W., Lindell, D. B., and Wetzstein, G., "Implicit Neural Representations with Periodic Activation Functions," , 2020. https://doi.org/10.48550/ARXIV.2006.09661.

[31] He, K., Zhang, X., Ren, S., and Sun, J., "Identity Mappings in Deep Residual Networks," , 2016. https://doi.org/10.48550/ARXIV.1603.05027.

[32] Pan, S., Brunton, S. L., and Kutz, J. N., "Neural Implicit Flow: a mesh-agnostic dimensionality reduction paradigm of spatio-temporal data," , 2022. https://doi.org/10.48550/ARXIV.2204.03216.

[33] Martel, J. N. P., Lindell, D. B., Lin, C. Z., Chan, E. R., Monteiro, M., and Wetzstein, G., "Acorn: adaptive coordinate networks for neural scene representation," *ACM Transactions on Graphics*, Vol. 40, No. 4, 2021, pp. 1–13. https://doi.org/10.1145/3450626.3459785.

[34] Yang, R., Xiao, T., Cheng, Y., Cao, Q., Qu, J., Suo, J., and Dai, Q., "SCI: A Spectrum Concentrated Implicit Neural Compression for Biomedical Data," , 2022. https://doi.org/10.48550/ARXIV.2209.15180.

[35] Yang, R., Xiao, T., Cheng, Y., Suo, J., and Dai, Q., "TINC: Tree-structured Implicit Neural Compression," , 2022. https://doi.org/10.48550/ARXIV.2211.06689.

[36] Botsch, M., *Polygon mesh processing*, A K Peters, 2010.

[37] Hauser, K., "Chapter 7: Representing Geometry, Robotic Systems (draft)," , 2025. URL https://motion.cs.illinois.edu/RoboticSystems, retrieved Jan 28, 2025.

[38] Zhou, K., Zhang, E., Bittner, J., and Wonka, P., "Visibility-driven Mesh Analysis and Visualization through Graph Cuts," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 14, No. 6, 2008, pp. 1667–1674. https://doi.org/10.1109/tvcg.2008.176.

[39] Jacobson, A., Kavan, L., and Sorkine-Hornung, O., "Robust inside-outside segmentation using generalized winding numbers," *ACM Transactions on Graphics*, Vol. 32, No. 4, 2013, pp. 1–12. https://doi.org/10.1145/2461912.2461916.

[40] Elfes, A., "Using occupancy grids for mobile robot perception and navigation," *Computer*, Vol. 22, No. 6, 1989, pp. 46–57.

[41] Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W., "OctoMap: an efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, Vol. 34, No. 3, 2013, pp. 189–206. https://doi.org/10.1007/s10514-012-9321-0.

[42] Tatarchenko, M., Dosovitskiy, A., and Brox, T., "Octree Generating Networks: Efficient Convolutional Architectures for High-Resolution 3D Outputs," *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[43] Jones, M., Baerentzen, J., and Sramek, M., "3D distance fields: a survey of techniques and applications," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 12, No. 4, 2006, pp. 581–599. https://doi.org/10.1109/TVCG.2006.56.

[44] Söderlund, H. H., Evans, A., and Akenine-Möller, T., "Ray tracing of signed distance function grids," *Journal of Computer Graphics Techniques Vol*, Vol. 11, No. 3, 2022.

[45] Park, J. J., Florence, P., Straub, J., Newcombe, R., and Lovegrove, S., "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2019, pp. 165–174. https://doi.org/10.1109/cvpr.2019.00025.

[46] Davies, T., Nowrouzezahrai, D., and Jacobson, A., "On the Effectiveness of Weight-Encoded Neural Implicit 3D Shapes," , 2020. https://doi.org/10.48550/ARXIV.2009.09808.

[47] Takikawa, T., Litalien, J., Yin, K., Kreis, K., Loop, C., Nowrouzezahrai, D., Jacobson, A., McGuire, M., and Fidler, S., "Neural Geometric Level of Detail: Real-time Rendering with Implicit 3D Shapes," , 2021. https://doi.org/10.48550/ARXIV.2101.10994.

[48] Rahaman, N., Baratin, A., Arpit, D., Draxler, F., Lin, M., Hamprecht, F. A., Bengio, Y., and Courville, A., "On the Spectral Bias of Neural Networks," 2018. https://doi.org/10.48550/ARXIV.1806.08734.

[49] Hart, J., "Sphere Tracing: Simple Robust Antialiased Rendering of Distance-Based Implicit Surfaces," 1995.

[50] Umesh, S., and Mittal, S., "A survey of techniques for intermittent computing," *Journal of Systems Architecture*, Vol. 112, 2021, p. 101859. https://doi.org/10.1016/j.sysarc.2020.101859.

[51] Singh, J., and Narayanan, P., "Real-Time Ray Tracing of Implicit Surfaces on the GPU," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 16, No. 2, 2010, pp. 261–272. https://doi.org/10.1109/tvcg.2009.41.

[52] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," , 2015. URL https://www.tensorflow.org/, software available from tensorflow.org.

[53] Chollet, F., et al., "Keras," https://keras.io, 2015.

[54] Dawson-Haggerty, M., et al., "Trimesh: A Python library for loading and using triangular meshes," https://trimsh.org/, 2024. Version 4.4.0, accessed 11 October 2025.

[55] Sullivan, C., and Kaszynski, A., "PyVista: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK)," *Journal of Open Source Software*, Vol. 4, No. 37, 2019, p. 1450. https://doi.org/10.21105/joss.01450.

[56] Kingma, D. P., and Ba, J., "Adam: A Method for Stochastic Optimization," , 2014. https://doi.org/10.48550/ARXIV.1412.6980.

[57] Agarap, A. F., "Deep Learning using Rectified Linear Units (ReLU)," , 2018. https://doi.org/10.48550/ARXIV.1803.08375.

[58] Ahrens, J. P., Geveci, B., and Law, C. C., "ParaView: An End-User Tool for Large-Data Visualization," *The Visualization Handbook*, 2005. URL https://api.semanticscholar.org/CorpusID:56558637.

[59] Zhou, Q., and Jacobson, A., "Thingi10K: A Dataset of 10,000 3D-Printing Models," , 2016. https://doi.org/10.48550/ARXIV.1605.04797.

[60] Zhou, Q.-Y., Park, J., and Koltun, V., "Open3D: A Modern Library for 3D Data Processing," *arXiv:1801.09847*, 2018.

[61] Liang, X., Zhao, K., Di, S., Li, S., Underwood, R., Gok, A. M., Tian, J., Deng, J., Calhoun, J. C., Tao, D., Chen, Z., and Cappello, F., "SZ3: A Modular Framework for Composing Prediction-Based Error-Bounded Lossy Compressors," , 2021. https://doi.org/10.48550/ARXIV.2111.02925.