Adaptive Multilinear Tensor Product Wavelets

Kenneth Weiss, *Member, IEEE*, and Peter Lindstrom, *Senior Member, IEEE*

Abstract—Many foundational visualization techniques including isosurfacing, direct volume rendering and texture mapping rely on piecewise multilinear interpolation over the cells of a mesh. However, there has not been much focus within the visualization community on techniques that efficiently generate and encode globally continuous functions defined by the union of multilinear cells. Wavelets provide a rich context for analyzing and processing complicated datasets. In this paper, we exploit adaptive regular refinement as a means of representing and evaluating functions described by a subset of their nonzero wavelet coefficients. We analyze the dependencies involved in the wavelet transform and describe how to generate and represent the coarsest adaptive mesh with nodal function values such that the inverse wavelet transform is exactly reproduced via simple interpolation (subdivision) over the mesh elements. This allows for an adaptive, sparse representation of the function with on-demand evaluation at any point in the domain. We focus on the popular wavelets formed by tensor products of linear B-splines, resulting in an adaptive, nonconforming but crack-free quadtree (2D) or octree (3D) mesh that allows reproducing globally continuous functions via multilinear interpolation over its cells.

Index Terms—Multilinear interpolation, adaptive wavelets, multiresolution models, octrees, continuous reconstruction.

1 Introduction

Multilinear interpolants are at the foundation of many key visualization techniques, including isosurfacing [10,27,31], direct volume rendering (DVR) [1] and texture mapping [21], which assume piecewise multilinear interpolants over a mesh's quadrilateral or hexahedral cells. However, despite their importance, there has not been much focus within the visualization community on techniques that efficiently generate and encode globally continuous (C^0) functions defined by the union of multilinear cells. Although this is straightforward for uniform meshes, it becomes more difficult on adaptive meshes generated by subsets of a regular grid, e.g. on meshes generated by quadtrees, octrees and kd-trees, which are becoming more common as the gulf between processing power and memory widens. Thus, we require well grounded approaches that efficiently incorporate adaptivity in domain and function space.

Wavelets provide a natural framework for such approaches and tensor product wavelets based on quadtrees and octrees have a rich history [34,40]. When derived from linear B-spline wavelets, they correspond to piecewise multilinear functions. To ensure continuity between grid cells of different resolution, such meshes are typically triangulated and the underlying multilinear function is *approximated* via linear interpolation on each triangle (2D) or tetrahedron (3D) [4, 20, 33, 47]. This increases the complexity of the mesh, and, more importantly, introduces directional biases into the mesh [8] and error in the function approximation. We present a novel approach that retains the piecewise multilinear nature of the function over an adaptive quadtree or octree of minimal size while enforcing C^0 continuity.

Despite the popularity of wavelet-based techniques for simplifying volumes, images and shapes, most of the techniques in the literature have approached the problem from the perspective of reducing the storage for encoding or transmission, e.g. via quantizing and thresholding the wavelet coefficients. Considerably less effort has gone into the development of data structures and evaluation techniques for representing and processing the function in a way that exploits its sparseness in the wavelet basis. Such sparseness may arise because the function is intrinsically smooth with few fine-scale details, or is approximated by discarding wavelets with small coefficients or with support outside a region of interest. In this case the function can be represented on a coarser, adaptively refined domain and later recovered *exactly* at full resolution on demand using the subdivision scheme associated with the wavelet scaling function. Such an adaptive representation is not only memory efficient, but can also be used to accelerate visualization tasks like ray casting by spatially varying the grid resolution.

In this paper, we introduce a framework for lifted, separable wavelet transforms over regular grids, in which we pay attention to the runtime representation of the adaptively refined domain. In particular, we discuss how to reconstruct, interpolate and visualize such adaptive functions through the use of a meshing scheme that organizes the wavelets into binary trees (1D), quadtrees (2D), and octrees (3D). Our results should be seen as complementary to the broad spectrum of work on offline encoding of wavelet coefficients [34, 36].

We derive adaptivity from the *dependency relations* among the tensor product wavelet lifting steps and mesh refinements. After reviewing lifting in the uniform case (Section 3), we formally analyze the dependency relations required for adaptive wavelet transforms (Section 4).

The main contributions of this work are:

- A unified framework for adaptive meshing of functions defined by an arbitrary subset of wavelet coefficients from a regular grid. Although this framework is general and applicable to wavelets with parametric synthesis scaling functions [39], in this paper, we focus on the popular linear B-spline tensor product wavelets [40] useful for visualization. We present simple rules for constructing the coarsest mesh that allows exact reproduction of the associated function. We treat both *interpolating* wavelets (i.e. those with just prediction) and *approximating* wavelets (i.e. those with prediction and updates) in a consistent manner.
- We show that the mesh can be further coarsened by replacing the multilinear interpolant with spectral interpolation [23]. Furthermore, we use this interpolant to highlight the correspondence between simple subsampling and interpolating linear B-spline wavelets, leading to a simplified subsampling strategy.
- Given an adaptive mesh defined by a subset of wavelet coefficients, we propose an efficient means of improving its approximation quality at no additional storage or reconstruction cost by *saturating* the mesh with all wavelets whose support are already present in the mesh.
- We describe a simple pointerless representation for quadtrees and octrees defined by scalar values at the vertices. Specifically, we only encode the vertices of the mesh, and can implicitly retrieve its cells, as well as its hierarchical and geometric relationships.

Our approach can be thought of as generalizing the ubiquitous interpolating linear wavelets defined on simplices within adaptive binary trees [15, 18, 49] to multilinear C^0 interpolating and approximating wavelets on adaptive quadtrees and octrees.

- *K. Weiss and P. Lindstrom are with Lawrence Livermore National Laboratory. E-mail: {kweiss,pl}@llnl.gov.*

Manuscript received 31 Mar. 2015; accepted 1 Aug. 2015; date of publication xx Aug. 2015; date of current version 25 Oct. 2015. For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

2 Background and related work

Multiresolution analysis has a rich history in the signal processing, computer graphics and CAD communities dating back several decades [28, 40]. The lifting scheme, which can be defined for any wavelet filter [13], was introduced by Sweldens [41] for biorthogonal wavelets, where the possibility of using an adaptive domain is hinted. Note that the term *adaptive* in the wavelet literature is often used in the context of data dependent wavelet transforms, where one can adapt the wavelet basis functions themselves to fit the data [11, 51]. While such an approach can fit within our framework, in this paper, we use the term to refer to adaptivity in the domain decomposition.

There are several similar approaches within the PDE community to adaptive grids [5, 9, 14, 29], most of which are similar to block-based AMR, in which a series of nested grids cover the region of interest. In [22], a set of samples with high wavelet magnitudes are retained which are used to interpolate missing samples. In this paper, we introduce a framework for reconstructing the smallest mesh to satisfy both interpolating and approximating wavelets. We therefore focus on efficient data structures to encode and reconstruct the sparse function.

Perhaps the most closely related work to ours is the adaptive approach of Linsen et al. [25], which derives an adaptive tetrahedral mesh from their progressive $\sqrt[n]{2}$ scheme [26] (based on the lifting approach of Bertram et al. [6]). Similar to other approaches [20, 33, 47], once the function is reconstructed, it is approximated in terms of the piecewise linear tetrahedra, rather than as a multilinear function. Specifically, they extract isosurfaces using Marching Tetrahedra [32] over the simplices. In contrast, our exact evaluation uses the underlying multilinear interpolant given by the wavelet basis. As we demonstrate in Section 6.5 for natural, scanned and simulation datasets, this can improve the fidelity of our approximation by several decibels on meshes defined by the same set of vertices.

Furthermore, our scheme is designed to include the full support of every included wavelet in our adaptive representation. This does not appear to be the case in [25], whose adaptive meshes are defined only by the dependencies induced by conforming simplex bisections, which do not generally include the entire support of the wavelets. We intend to examine this issue more fully in a forthcoming paper.

Another related approach is CHARMS [19], which uses the two-scale relation of arbitrary nested basis functions to simplify adaptivity in Finite Element spaces. This is similar to our 'Full internal' stencil in Section 5.2.2. We reduce the storage requirements through the use of the lifting scheme and through our supercube encoding (see Section 5).

There has been a lot of work in using multiresolution and adaptive grids for processing scalar functions defined over regular grids. A d-dimensional octree is a hierarchical domain decomposition based on the regular refinement of cubes. To simplify our discussion throughout this paper, we use the terms *octree* and *cubes* to generically refer to d-dimensional octrees defined by d-cubes, where d is the dimension of the domain, and we specify the dimension only when necessary.

Under *regular refinement*, a d-cube $\mathscr{C}$ is replaced by 2^d similar d-cubes. This operation defines a nested containment hierarchy in which the 2^d cells created during the regular refinement of a d-cube $\mathscr{C}$ are its *children*, and $\mathscr{C}$ is their *parent*. The vertices of the children are defined by the midpoints of the parent cube's *faces* (e.g. edges, facets), and each child contains one of the vertices of $\mathscr{C}$ as well as its midpoint.

The octree's *root* covers the entire domain. Cubes that are not refined have no children and are referred to as *leaf cells* of the octree. The *level* of a cube $\mathscr{C}$ in an octree T is the number of refinements necessary to obtain $\mathscr{C}$ from T's root. In a *uniform octree*, all leaf cells are at the same level, otherwise, the octree is *adaptive*. A *balanced* octree is one in which neighboring cells differ by at most one level. Specifically, an octree is *edge-balanced* if each edge in the octree is on the boundary of d-cubes from at most two consecutive levels.

Consider the regular cubical mesh defined by a uniform octree T. We refer to the i-dimensional faces of T as *primal cubes*, and to the axis-aligned $(d-i)$-cubes defined by connecting the centers of its adjacent d-cubes as *dual cubes*. Figure 1 illustrates a 1-cube (edge) in 3D and its dual 2-cube.

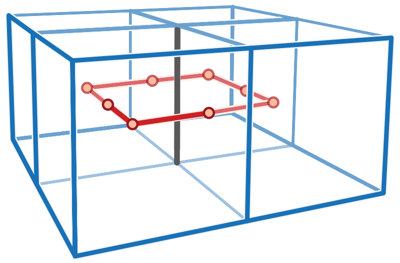


Fig. 1. An edge (black) and its dual 2-cube (red) in a hexahedral mesh.

Most octree representations have focused on encoding data with the leaves of the tree [35], which is sufficient when the data is cell-centered. However, when the data is associated with the vertices of the mesh, this can lead to a redundant encoding of the vertex data. Our representation utilizes a pointerless supercube representation [46] originally proposed for diamond hierarchies [49]. The encoding of the vertex data can be seen as a non-redundant tile-based representation of the vertex data (similar to [2, 17], among others). An interesting aspect of this representation is that leaf cells can be determined implicitly from the set of encoded vertices, and, thus, do not need to be encoded.

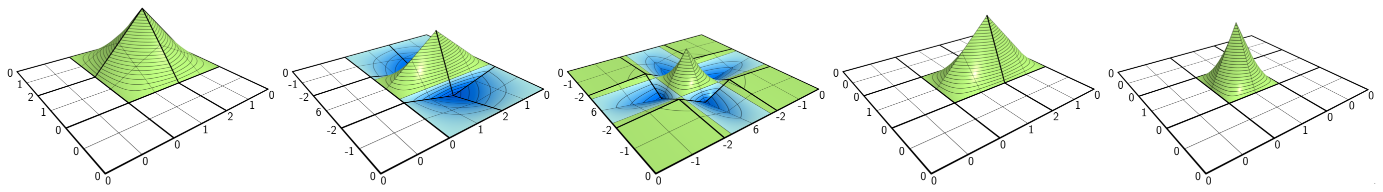
Applications such as surface extraction and ray tracing typically require a function with at least C^0 continuity (i.e. crack-free). However, adaptive octree meshes are not conforming. There are several approaches to achieve continuity from an adaptive octree mesh, including explicitly patching cracks in the extracted surface [38], modifying the field values at the vertices [30], or making the primal [4, 20, 35, 47] or dual [16, 44] domain conforming using noncubical cells. Although this is typically implemented by triangulating the lower resolution neighbors on a balanced octree, we exploit the fact that under a multilinear interpolant the function varies linearly along the edges of the mesh cubes. Thus, by locally refining some edges using linear interpolation of their midpoints, we obtain a piecewise multilinear C^0 interpolant over cubical cells despite the presence of T-junctions.

3 Lifting for uniform tensor product wavelets

In this section, we anticipate the discussion of adaptively refined tensor product wavelets by first discussing the uniform case. We assume our domain is a d-dimensional uniform grid $\mathscr{G}$ of resolution $(2^L+1)^d$.

The lifting scheme is a means of factoring wavelet transforms into simpler elementary operators. The forward lifting transform (wavelet analysis) consists of three steps. It first splits the vertices V into two sets V_w and V_s. V_w is associated with wavelets at the current level of resolution, whereas the remaining set V_s is associated with *scaling functions*. A *w-lift* generates wavelet coefficients for the vertices V_w. Each coefficient is the residual of a prediction P formed by a weighted sum of function values at neighboring vertices from the set V_s. An *s-lift* update is then optionally applied to the vertices V_s as a weighted sum U of neighboring wavelet coefficients. This step exists to ensure that the mean and possibly higher order moments are preserved in the lower resolution signal represented by V_s. Multiple resolutions are obtained by recursively applying this transform to the remaining vertices V_s. The inverse lifting transform (wavelet synthesis) is trivially defined by reversing this sequence of steps.

When applied to a mesh, lifting can be understood in the context of regular refinement, where the splitting phase modifies the connectivity of the mesh, while the s-lift and w-lift phases modify attributes associated with the mesh (e.g. scalar values). Consider a regularly refined binary tree T, i.e., a uniform 1D octree, whose edges are cells at level ℓ defined over a regular grid containing $2^\ell+1$ vertices. Then the inverse transform uniformly refines T to level $\ell+1$ by inserting a new vertex at the midpoint of each edge of T, converting T to a uniform octree at level $\ell+1$. The inverse s-lift and w-lift stages modify the values of T's vertices (nodes) as well as the newly inserted midpoints, respectively.



(a) Scaling function $\phi_x\phi_y$ (b) Edge wavelet $\phi_x\widetilde{\psi}_y$ (c) Square wavelet $\widetilde{\psi}_x\widetilde{\psi}_y$ (d) Edge wavelet $\phi_x\overline{\psi}_y$ (e) Square wavelet $\overline{\psi}_x\overline{\psi}_y$

Fig. 2. 2D basis functions for approximating (b, c) and interpolating (d, e) bilinear B-spline wavelets. With respect to the embedding 3×3 grid (bold), scaling functions (a) are associated with vertices, while wavelets are associated with edges (b, d) and squares (c, e).

3.1 Interpolating and approximating 1D wavelets

We consider two simple lifted wavelet transforms. *Interpolating wavelets* use only a prediction step (w-lift), leaving the function values at V_s unchanged from one level to the next. *Approximating wavelets* have both a prediction (w-lift) and an update (s-lift) step. Thus, interpolating wavelets have simpler computational requirements and smaller support while approximating wavelets can better preserve properties of the function (such as its mean).

To make this more concrete, consider the biorthogonal linear B-spline wavelets [12,40] defined on a uniform 1D grid, where all vertices are connected to their adjacent vertices via an edge. In the w-lift, we predict the value of each odd vertex $v_{2i+1} \in V_w$ as the average of its two even neighbors: $P(v_{2i+1}) = \frac{1}{2}\left[f(v_{2i}) + f(v_{2i+2})\right]$. In the s-lift, we update the values of the even vertices V_s by adding to them $U(v_{2i}) = \frac{1}{4}\left[\hat{f}(v_{2i-1}) + \hat{f}(v_{2i+1})\right]$, where $\hat{f}(v) = f(v) - P(v)$ denotes a wavelet coefficient resulting from the forward w-lift. This lifting scheme results in corresponding stencils for the approximating synthesis wavelet $\widetilde{\psi} = \frac{1}{8}[0,-1,-2,6,-2,-1,0]$ and interpolating wavelet $\overline{\psi} = [0,1,0]$. Both transforms have $\phi = \frac{1}{2}[0,1,2,1,0]$ as their scaling functions, implying a piecewise linear subdivision in 1D.

3.2 Tensor product wavelets

The 1D lifting scheme described above can be easily generalized to higher dimensions using a separable tensor product of 1D wavelets [40]. In this approach, we cyclically apply the 1D scheme to the axis aligned samples along each individual dimension, leading to a tensor product of i wavelets ψ and $(d-i)$ scaling functions ϕ.

The correspondence between lifted wavelets and regular refinement in 1D can be generalized to tensor product wavelets. Specifically, a basis function defined by the tensor product of i wavelets ψ and $(d-i)$ scaling functions ϕ is associated with the midpoint of an i-cube in the d-dimensional grid (which is also the midpoint of a dual $(d-i)$-cube; see Figure 1). Thus, in 2D, the tensor product scaling functions $\phi_x\phi_y$ are associated with the vertices of the grid, while the wavelets $\phi_x\psi_y$ and $\psi_x\phi_y$ are associated with the midpoints of edges, and the wavelets $\psi_x\psi_y$ are associated with midpoints of the square cells. Since the scaling functions for linear B-spline wavelets correspond to linear interpolation, their tensor product corresponds to multilinear interpolation, namely, bilinear interpolation in 2D and trilinear interpolation in 3D.

Figure 2 shows the scaling function and two types of interpolating and approximating linear B-spline tensor product wavelets, along with their support on a refined 3×3 grid. Because the scaling function is piecewise bilinear, the wavelets correspond to a set of bilinear patches at the finer level or resolution (this is referred to as the *two-scale relation* [28,40]). Note also that the stencils (i.e. the values at the vertices of the refined 3×3 grid) are defined by the tensor product of the 1D stencils (see Section 3.1).

3.3 Filtered wavelet coefficients

A typical workflow for using the wavelet transform consists of transforming the function into the wavelet domain, analyzing and modifying the function in the wavelet domain, and finally converting the modified function back to the spatial domain. During the analysis phase, the coefficients are often filtered or quantized in an application dependent way, e.g. to remove noisy details, or for efficient storage and transmission. These filters can be based on properties of the wavelet, such as its coefficient's magnitude or the range of isovalues in its support. Hierarchical (e.g. refinement level) and spatial information can also be used, e.g. for region of interest (ROI) queries.

Although the function has been simplified, the domain is the same, and, thus, requires the same amount of processing and storage at runtime. In the following section, we propose a refinement strategy that adapts the underlying complexity of the reconstructed function.

4 Adaptive Tensor Product Wavelets

Our adaptive refinement scheme is applicable to tensor product wavelets whose scaling functions can be directly evaluated from a stencil (as in [39]). We utilize a regularly refined octree to organize our adaptive mesh. For each wavelet coefficient that we wish to include in our function, we ensure that its corresponding wavelet can be fully reconstructed in the resulting mesh. After running our (adaptive) inverse lifting transform, we use the wavelet scheme's interpolant to evaluate the function at any point in the domain. Thus, our results exactly match that of the function on the full domain when reconstructed from the same set of wavelet coefficients. For linear B-spline tensor product wavelets, the associated interpolant is multilinear and smooth over each cell of the octree, and is C^0 continuous across cell boundaries.

4.1 Reduced neighborhoods for wavelet support

As mentioned above, a common reconstruction strategy is to either use the entire full-resolution domain to reconstruct the function from a subset of the wavelets [28,40], or to use a set of (sufficiently large) nested uniform grids surrounding each wavelet, as is typical in the AMR community [5,14]. Our goal is to extract the coarsest mesh that faithfully reconstructs a given (arbitrary) set of wavelets. We achieve this by exploiting properties of regular refinement, of our interpolant, and of the stencil's coefficients to reduce the number of vertices (and cells) required to reconstruct each wavelet.

Recall that the *two-scale relationship* implies that wavelets and scaling functions at level ℓ are defined as linear combinations of scaling functions at level $\ell+1$. Thus, we can exactly reconstruct a given wavelet ψ if our mesh includes all vertices at level $\ell+1$ associated with the scaling functions defining its two-scale relationship. For tensor product wavelets, these vertices lie within a rectilinear d-dimensional stencil of coefficients surrounding the wavelet's associated vertex, and the stencil values are determined by the tensor product of its 1D stencils, as described in Section 3. For linear B-splines, these stencils cover five, three and seven grid points at level $\ell+1$ for scaling function ϕ, interpolating wavelet $\overline{\psi}$ and approximating wavelet $\widetilde{\psi}$, respectively. For example, the 2D approximating wavelet $\widetilde{\psi}_x\widetilde{\psi}_y$ over a square cell (see Figure 2c) is defined as the weighted sum of the $7\times7=49$ scaling functions within its stencil, and a wavelet $\widetilde{\psi}_x\widetilde{\psi}_y\widetilde{\psi}_z$ over a cubic cell is defined by $7^3=343$ scaling functions. Our discussion below examines these scaling functions in terms of the set of grid points at level ℓ, which comprise the vertices bounding the cells in the support of each wavelet, and those at level $\ell+1$ bounding their refined cells. In all cases, we require the full set of grid points from the former group but not necessarily all grid points from the latter group. For (1D) linear B-splines, the support of the scaling function ϕ covers two cells at level ℓ defined by three level-ℓ grid points, while $\overline{\psi}$ and $\widetilde{\psi}$ have supports covering one and three level-ℓ cells, respectively.

4.1.1 Full internal stencil support

We first observe that the boundaries of the stencils have a layer of zeros, and that these are all aligned with the boundaries of cubic cells from level ℓ. Since the values of grid points at level $\ell+1$ (i.e. the faces of the cubic cells) can be trivially interpolated from their zero-valued boundary vertices, we can safely omit them.

Note that for interpolating linear B-spline wavelets $\bar{\psi}$, all that remains is an inner subgrid centered around $\bar{\psi}$, with three grid points along directions defined by scaling functions, and one grid point along wavelet directions. Each such stencil point corresponds to a face of $\bar{\psi}$'s corresponding dual cube (see Figure 3a).

In our running example, eliminating the boundary grid points reduces the samples required for reconstructing $\widetilde{\psi}_x\widetilde{\psi}_y$ to 37 (i.e. we have removed the twelve edge midpoints on the boundary of its support).

4.1.2 Exploiting properties of the scaling function

Although the previous step reduces the required support of each wavelet, it still includes many vertices whose values can be recovered from our wavelet interpolant. We can exploit properties of our interpolant to remove stencil vertices whose values can be interpolated directly from other samples (e.g. from the vertices of level-ℓ cubes). Specifically, for the 1D linear B-spline wavelet $\widetilde{\psi}$, the collinearity of $[0,-1,-2]$ in the approximating wavelet stencil allows us to exclude filter weight -1 as it is the linear average of 0 and -2. This allows the wavelet to be represented using cells of varying resolution.

In fact, we can decompose $\widetilde{\psi}$ into the sum of two scaling functions ϕ associated with the internal cell vertices of its stencil and a single interpolating wavelet $\bar{\psi}$ associated with the midpoint of its internal edge. This analysis can be extended to approximating multilinear B-spline wavelets. Let ψ be a wavelet associated with a k-cube $\mathscr{C}$ in the mesh (i.e. it is located at the midpoint of $\mathscr{C}$, where $k \leq d$). Then ψ can be decomposed into 2^k scaling functions associated with the vertices of $\mathscr{C}$ and into the sum of interpolating tensor product wavelets associated with each remaining face of $\mathscr{C}$. For example, the 2D approximating square wavelet associated with the midpoint of the center square of the grid (Figure 2c) can be decomposed into four scaling functions associated with its vertices, four interpolating edge wavelets and a single interpolating square wavelet. The advantage of this decomposition is that we only need to include the stencil points associated with the five interpolating wavelets. In our running example, this reduces the number of samples required for reconstructing $\widetilde{\psi}_x\widetilde{\psi}_y$ to 25, i.e. the sixteen grid points at level ℓ, the midpoints of the four edges of the center square and the midpoints of their five incident squares at level $\ell+1$.

4.1.3 Using higher order predictors

We can further reduce the overhead of each wavelet's stencil by incorporating a higher order predictor into our reconstruction scheme.

In the case of lifted interpolating linear B-spline wavelets it can be shown that the underlying predictor at the midpoint of a k-cube $\mathscr{C}$ is not given by a multilinear average of its corners, but rather by the "radial" spectral predictor [23] on the 3^k stencil that incorporates all faces of $\mathscr{C}$. This predictor is given by the weighted average of 3^k-1 points:

$$P = -\sum_{j=1}^{k}\left(-\frac{1}{2}\right)^j \sum_i f(v_i^{k-j})$$

where v_i^{k-j} denotes the i^{th} vertex of the $(k-j)$-cube centers of the faces of $\mathscr{C}$. Our interpolating wavelet predictor is equivalent to applying this predictor successively to edges, facets, and cubes, using any known values and spectrally predicting unknown values. It is easy to verify that when all vertices other than cube corners are multilinearly interpolated, P reduces to multilinear interpolation. Furthermore, since radial interpolation only changes the values at the midpoints of a cube's faces, each such patch can be seen as a set of multilinear patches at the next level of resolution. Applying the spectral radial predictor to a multilinear B-spline wavelet reduces the required stencil to just the grid points at level ℓ and the faces of primal cube $\mathscr{C}$ associated with the wavelet. Concluding our running example, a radially predicted $\widetilde{\psi}_x\widetilde{\psi}_y$ requires only 21 grid points (i.e. the 16 grid points at level ℓ, and the center and four edge midpoints on the internal square).

A note about radial prediction for approximating wavelets: Radial prediction works as described for interpolating wavelets, and for approximating wavelets that do not have additional internal stencil grid points. However, due our implementation using a separable lifting transform, radial prediction can potentially introduce some interpolation errors when only a subset of the internal stencil grid points are present, as some s-lift updates might not reach these vertices if the vertices required by the lifting scheme are missing. As a compromise, in our current implementation (which cyclically lifts along the x, then y, then z axes), we conservatively retain all stencil grid points that lie in the same x-plane as the wavelet (in 2D) or in the same xy-plane (in 3D), as this guarantees the stencils of s-lift updates to all vertices. An alternative data-dependent approach, which we leave for future work, might be to further analyze lifting scheme's dependencies and only add the vertices that are necessary to correctly reconstruct the wavelet.

5 Octree representation and function evaluation

In this section, we discuss our pointerless *nodal octree*, and describe how to insert wavelets using the various adaptive stencil supports discussed in Section 4.1 and how to evaluate the underlying field.

5.1 Representation

Our implicit octree structure encodes only the (adaptive subset of) vertices and their associated scalar values. Because the distribution of vertices in the mesh exhibits significant spatial coherence, we reduce the overhead due to specifying vertex coordinates using an efficient supercube representation [46], which clusters (up to) 4^d-2^d vertices (12 vertices in 2D; 56 vertices in 3D). The vertices present in the adaptive octree mesh are indicated using one bit per vertex, resulting in a per-supercube overhead of 7 bytes for the vertex flags and 6 bytes for supercube coordinates in 3D. The nodal function values are stored in tiles of size 4^d-2^d and the sets of supercubes at each level of resolution are stored in a hash map, providing random access to the mesh vertices and their associated scalar values. The mapping from grid vertices to supercubes involves only simple bit manipulations [46]. To traverse the tree, we only need to check if the midpoint of the current cube is present in the mesh, in which case the cube is not a leaf.

5.2 Wavelet stencil implementation

In Section 4.1, we described the characteristics of the different optimized wavelet stencils for interpolating and approximating tensor product B-spline wavelets. Here, we discuss how they can be efficiently implemented within our (pointerless) d-dimensional octree data structure. Our wavelet stencils are implemented in terms of (a) iterators of the faces of primal and dual cubes of a regular grid, and (b) standard octree refinement. The former are all implicitly determined from the coordinates of the input point p, corresponding to the center of a k-dimensional (primal) cube and its $(d-k)$-dimensional dual cube [45].

For the latter, recall that every vertex in the domain uniquely corresponds to a single k-cube in the octree, i.e. it is the center of a k-cube at some level in the hierarchy. Assume we have an octree T, to which we would like to add a wavelet ψ, whose support is centered at vertex p on level ℓ of T. Our strategy is given in two conceptual steps, although they can be implemented concurrently. First, we set up the proper *hierarchical context* for ψ's stencil by refining some octree blocks in the neighborhood of p (see Section 5.2.1). This takes care of all the octree refinement for the wavelet's stencil up to level ℓ, and is implemented as a set of standard octree refinements $\textsc{OctreeRefine}(\mathscr{C})$ for a d-dimensional cube $\mathscr{C}$. This function (recursively) checks that $\textsc{OctreeParent}(\mathscr{C})$ is refined and then refines $\mathscr{C}$. During the refinement, we insert vertices at the midpoints of all faces of the refining cube $\mathscr{C}$. After this step, all other modifications can be done locally without triggering additional octree refinements. Our second step is to insert the set of vertices at level $\ell+1$ comprising the stencil's internal vertices (see Section 5.2.2).

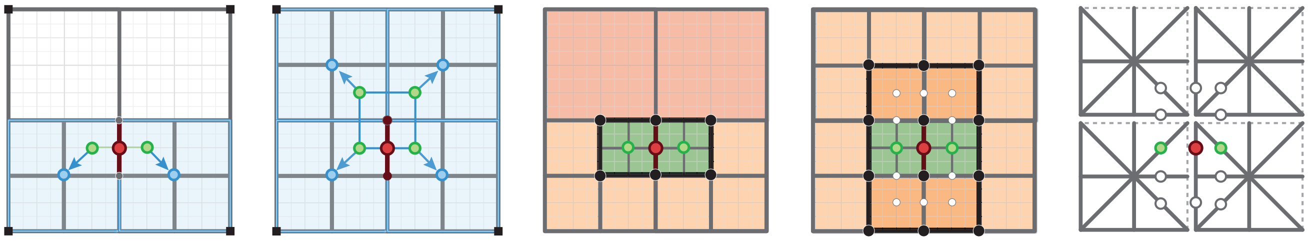

(a) $\bar{\psi}$ dependencies
(b) $\tilde{\psi}$ dependencies
(c) $\bar{\psi}$ stencils
(d) $\tilde{\psi}$ stencils
(e) Supercube vertices

Fig. 3. Hierarchical dependencies (a, b), stencil vertices (c, d) and supercube encoding (e) for interpolating $\bar{\psi}$ (a, c) and approximating $\tilde{\psi}$ (b, d) edge wavelets (red) in a quadtree. (a, b) Wavelets depend on the octree parents (blue) of the vertices of a dual cube (green) of a single vertex from the mesh (the wavelet itself in (a) and that of its upper endpoint in (b)). (c, d) Wavelet stencils require all vertices from the previous level of resolution (black vertices) within the wavelet's support (black edges, compare to Figure 2). Additional vertices are required for radial (red), multilinear (red and green) and full internal (red, green and white). (e) Vertices associated with the supercubes [45] at the wavelet's level of resolution.

5.2.1 Hierarchical dependencies

Since octree refinements apply to d-cubes, and our wavelet's midpoint is, in general, associated with a k-cube, we must first find a set of d-cubes in its neighborhood whose refinement will give us the right context. Recall from Section 2 that the vertices of the dual mesh correspond to the d-cubes of the primal mesh. The specifics depend on whether we are working with interpolating or approximating wavelets.

The support of an *interpolating* linear B-spline wavelet is entirely within the primal d-cubes associated with its $(d-k)$-dimensional dual-cube (Figure 3a). Thus, to create these cubes, we only need to apply OCTREEREFINE to the OCTREEPARENT of the dual cube vertices.

In the *approximating* case, a similar trick works, but rather than finding the dual cube associated with the wavelet's primal cube, we need to find the dual cubes associated with the vertices of its primal cube. This follows from the observation that the support of the approximating wavelets lie within a $3^k 2^{d-k}$ grid of cubes at level ℓ, where the dimensions that are three cubes wide are tangent to its primal dimensions (see Figures 2 and 3b). However, by considering the pattern of nested octree refinement, we observe that the sets of cubes generated by this process are nested. That is, the octree parents of one of the vertex's dual cubes is a superset of those of all the others, which we can safely ignore. Specifically, we want the vertex of the primal cube that is furthest from the midpoint of the supercube containing p (the vertex at the upper endpoint of the wavelet's edge in Figure 3b). This can easily be calculated by a few bit manipulations on its coordinates [46].

In summary, for each wavelet ψ, we identify a single vertex v (either its midpoint, in the case of interpolating wavelets, or one of its vertices, in the case of approximating wavelets), and for each vertex v_j of the dual cube of v, we apply OCTREEREFINE(OCTREEPARENT(v_j)).

5.2.2 Interior wavelet stencils

Given the proper octree context set up in the previous section, we can now implement the specific stencil described in Section 4.1.

Full rectilinear stencil Here, we simply add to T all vertices in the full subgrid of dimensions $M^k N^{d-k}$ centered at p, which has M grid points in dimensions that are primal and N in dimensions that are dual. For approximating wavelets, M is 7 and N is 5; for interpolating wavelets, the values are 3 and 5, respectively.

Full internal Since we already have the grid points at level ℓ, we only need to add those at level $\ell+1$. We can therefore use the same procedure as in the above step, after setting $M=5, N=3$ for approximating, and $M=1, N=3$ for interpolating.

Min multilinear For proper multilinear interpolation, we need to add all face centers of the dual cube surrounding each edge of p's primal cube. We implement this through iterators on the primal cube edges, which call iterators on the associate dual cube faces. For interpolating wavelets, *'Min multilinear'* and *'Full internal'* are equivalent.

Min radial For the simplest radially interpolated stencil, we add the midpoints of the faces of the primal cube. This is sufficient for the interpolating wavelets where there are no data dependencies due to s-lifts. For our approximating wavelets, we take a conservative approach and add all vertices that can have potential data dependency conflicts. Generally speaking, since we cyclically apply our lifting steps, this includes the vertices added in the *'Min multilinear'* step above that have the same y-coordinate (2D) or z-coordinate (3D) as ψ's midpoint.

5.3 Function evaluation

We have the option of either storing the wavelet coefficients or inverse-transformed function values with the octree vertices. In the former case, the lifted inverse transform is executed by traversing the octree breadth-first level by level and applying standard 1D lifting operations to transform the wavelet coefficients. Since each wavelet has the proper stencil for lifting (see Section 4), we compute the same value at the vertices of the mesh as we would in the uniform case.

To evaluate the function at a single point, we traverse the octree from the root and follow the branch containing the point until we reach a leaf cube. Given function values at the corners of this cube, we then apply multilinear or radial interpolation, depending on our extracted stencil. When we wish to evaluate the function over the entire domain, e.g. to display an image, we simply visit all leaf cubes and use our interpolant to compute the grid point values.

6 APPLICATIONS AND RESULTS

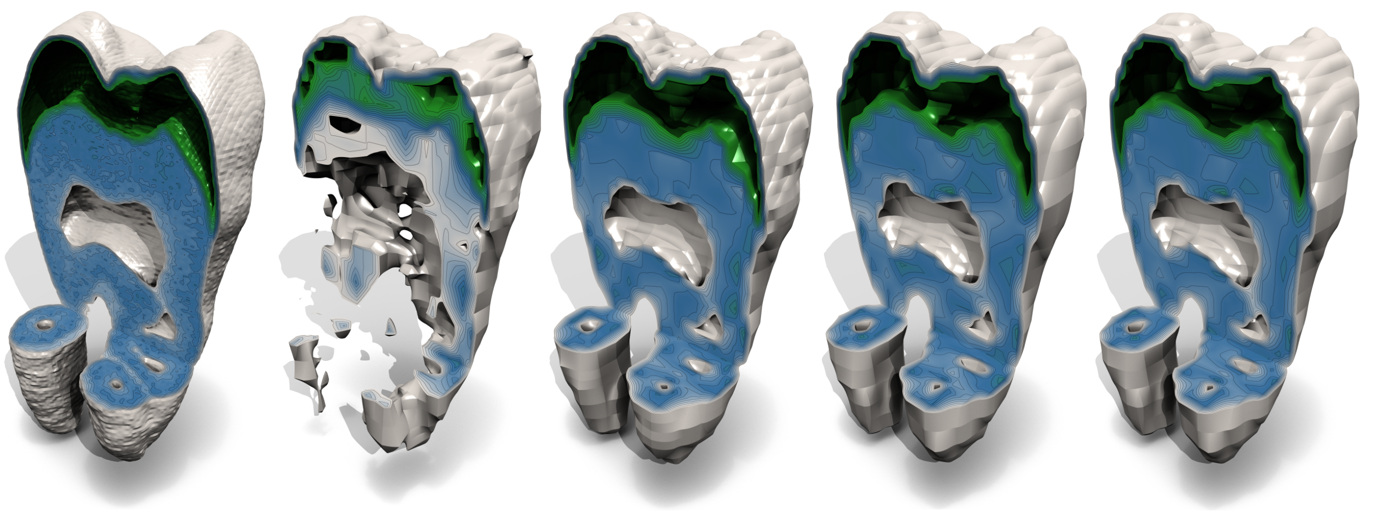
In this section, we discuss the utility and efficiency of our adaptive approach against the simpler uniform case. We also compare several applications of interpolating and approximating wavelets and consider the benefits of using the underlying multilinear interpolant over the adaptive cubical mesh rather than a linear interpolant over a triangulated domain. Due to space considerations, we provide a comprehensive analysis over a range of datasets in a supplementary appendix.

In the following, we denote approximating wavelets as a, and interpolating wavelets as i. We use superscripts to denote features of the wavelets (adaptivity, *normalization* and *saturation*, as defined below), and subscripts to denote the interpolation and choice of stencil. Thus, a_m^U denotes a mesh reconstructed using an approximating wavelet and multilinear (m) interpolation at uniform resolution (U). We compare the fidelity of the reconstruction in terms of the logarithmically scaled Peak Signal to Noise Ratio (PSNR) [43].

6.1 Uniform meshes and progressive refinement

Our approach allows us to build a mesh by selecting any subset of the wavelets in the domain, so a natural first step is to filter the wavelets by hierarchy depth. The uniform lifting scheme applied to tensor product wavelets enables a straightforward progressive level of detail (LOD) extraction on a reduced domain at the granularity of one mesh per level of resolution. That is, we can reconstruct the mesh on a uniform octree of maximum level $\ell \leq L$ by applying a full cycle of lifting steps.

Our approach enables selection of all wavelets at sublevel granularity (which we refer to as depths). Thus, we can select all wavelets up



(a) a_m^U $(17M,17M,316K)$ Full resolution (b) a_m^I $(5K,64K,3.6K)$ $PSNR$:24.5 (c) a_m^{II} $(45K,64K,3.6K)$ $PSNR$:34.0 (d) $a_m^{I,n}$ $(5K,21K,1.2K)$ $PSNR$:35.4 (e) $a_m^{II,n}$ $(14K,21K,1.2K)$ $PSNR$:35.9

Fig. 4. Interval volume renderings comparing wavelet saturation and normalization in adaptive meshes approximating the Tooth dataset (a) from its highest 0.03% wavelet coefficients. The unnormalized, unsaturated mesh (b) uses a significant portion of its vertex budget on high resolution noise (not pictured), and fails to capture the shape of the tooth. Its saturated counterpart (c) successfully captures the shape, while both normalized meshes (unsaturated (d) and saturated (e)) achieve better approximations using a considerably smaller mesh. The numbers in the subcaptions indicate the number of wavelet coefficients, vertices and supercubes in the extracted mesh, as well its Peak Signal to Noise Ratio (PSNR).

to level ℓ, and then also add all the level-ℓ edge wavelets, followed by all the square wavelets (followed by the cube wavelets in 3D).

Note that the tensor product dependencies imply that we insert the depths in order of increasing dimensions (i.e. edges, then squares, then cubes, ...), while in the RSB-based $\sqrt[n]{(2)}$ scheme of Linsen et al. [26], the depths are inserted in the reverse order (i.e. cube centers followed by square centers and then edge centers). We have observed that most of the *power* within each level of resolution lies within the edge wavelets (see the dotted lines in Figure 9, where adding the edge-centered wavelets significantly impacts the PSNR, but adding the face-centered and cube-centered wavelets has much less impact).

An interesting result derived from our study on adaptive stencils (Section 4.1) is that, while we are free to select the wavelets associated with the edges of the mesh, giving us depth-based adaptivity in the function space, this does not give us any additional adaptivity in the corresponding mesh. That is, once we incorporate the edge wavelets at level ℓ, our stencil completion algorithm (Section 5.2) will add all other vertices at level ℓ, which are required by the adaptive lifting algorithm.

6.2 Wavelet magnitude thresholding

The next natural step is to use a predicate based on wavelet magnitude to filter the wavelets composing our approximated function.

Normalization. To minimize the approximation error for a given wavelet coefficient budget, the coefficients $\{\hat{f}\}$ may be selected by magnitude. However, the level of resolution and centering on edges, facets, and cubes of each wavelet basis function dictate its norm $\|\psi\|$, and hence influence the mean square error. For orthogonal wavelets, each excluded wavelet incurs an error of $\hat{f}^2\|\psi\|^2$, suggesting that we should select wavelets in order of decreasing $\|\hat{f}\psi\|$ (equivalent to normalizing the basis functions). Contrary to earlier work [20], we found that such normalization can have a significant impact on the quality of the results for a given mesh budget. Specifically, as we demonstrate below, normalization tends to promote the coarser details of the function, while avoiding higher resolution details, such as textures.

Saturation. To ensure that large-scale features are present, we may optionally include coefficients for all wavelets whose stencil is already present in the adaptive mesh (with respect to our chosen stencil scheme; see Section 4.1). This increases the number of wavelet coefficients while keeping the number of vertices, and thus, the mesh representation, fixed. We refer to this saturated selection strategy as Type II, with Type I denoting the original unsaturated set of coefficients.

Table 1. Numbers of required vertices in the support of approximating (a) and interpolating (i) wavelet stencils. Starting from the rectilinear stencil $(m, grid)$, we can remove the mulitlinearly interpolable vertices on the support boundaries (m, int), and on the interior (m, min). The radial interpolant allows us to remove more vertices (r, opt), but depending on the stencil's orientation, we conservatively include some additional vertices $(r, slice)$. Note that $i_m := i_{m,int} (= i_{m,min})$ and $i_r := i_{r,slice} (= i_{r,opt})$.

Stencil	Edge$_{1D}$	Square$_{2D}$	Edge$_{2D}$	Cube$_{3D}$	Square$_{2D}$	Edge$_{3D}$
$a_{m,grid}$	7	49	35	343	245	175
$a_{m,int}$	7	37	25	181	119	79
$a_{m,min}$	5	25	15	125	75	45
$a_{r,slice}$	5	23	{13,15}	99	{55,65}	{37,39,45}
$a_{r,opt}$	5	21	13	83	53	37
$i_{m,grid}$	3	9	15	27	45	75
i_m	3	9	9	27	19	27
i_r	3	9	7	27	17	19

Figure 4 compares the four combinations of wavelet normalization and saturation on the Tooth dataset [42] initialized by thresholding the top 0.03% of approximating wavelet coefficients. It is apparent that the unsaturated and unnormalized mesh a_m^I in 4(b) wastes much of its wavelet budget on high frequency noise, and does not faithfully reconstruct the shape of the tooth. At the same time, these wavelets can be relatively deep in the hierarchy, and therefore, their stencils impose larger hierarchical overhead. Since many of the wavelets associated with these extra vertices are fully supported in the mesh, we achieve a more faithful approximation by saturating the mesh (a_m^{II} in 4(c)), yielding a significant improvement in PSNR. By thresholding on normalized wavelets, we improve the approximation quality (e.g. increased PSNR), and considerably reduce the footprint of the mesh (by $\approx 2/3$). This is in line with our experiments on a range of 2D and 3D datasets over a set of extracted meshes. See the supplemental appendix, where we compare the effects of saturation, normalization and interpolation against the number of wavelets, vertices and supercubes and PSNR.

6.3 Efficiency of supercube-based octree representation

Here, we consider the efficiency of our adaptive representation in terms of wavelet selectivity, the overhead of stencil size, and our supercube-based encoding for the extracted meshes.

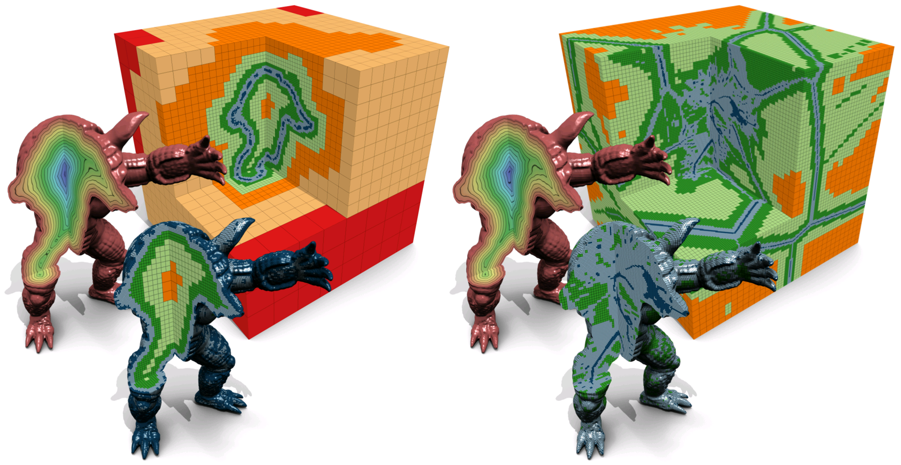
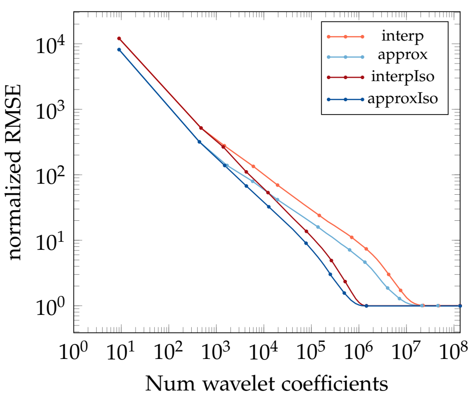



Fig. 5. Half-open interval volumes extracted from the Armadillo distance field. Comparison of selecting wavelets intersecting the zero level set of a 513^3 distance field (left) and selecting only by magnitude (middle). Saturated (Type II) normalized approximating wavelets $a^{II,n}$ were used in both cases. The images reveal the interior distance field (black contours of cutaway armadillo meshes) and octree depth (colored cubes of cutaway armadillo meshes and of cutaway octrees). Notice the distance field singularities preserved (middle), which do not contribute to the quality of the extracted surface. The relative distance error and number of wavelets, vertices, and supercubes are (1.23; 677 K; 1.27 M; 54.0 K) on the left and (3.67; 1.84 M; 3.39 M; 168 K) in the middle. The chart on the right plots the deviation from zero (normalized Root Mean Squared Error) given by the approximate fields for 1,000,000 samples from the original triangulated surface.

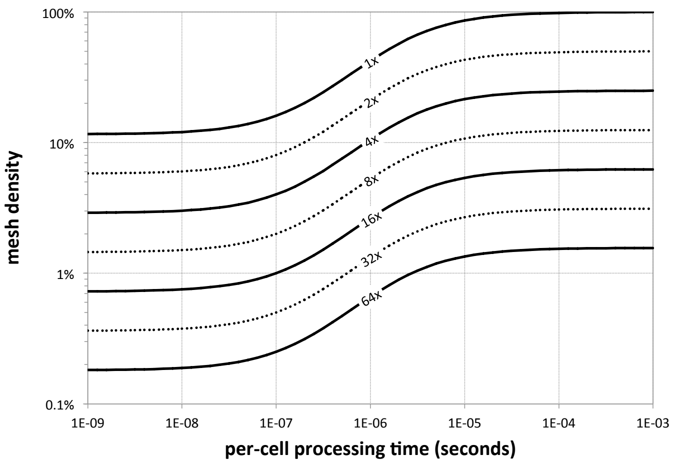


Fig. 6. Expected speedup (contours) due to adaptive mesh relative to full resolution datasets for a given cell processing time in our visualization kernel (x-axis) and density in our adaptive representation (y-axis).

Table 1 lists the overhead per wavelet for each type of stencil defined in Section 4.1, in terms of the number of required vertices. Simply removing the boundary vertices (m, int) of the wavelet stencil significantly reduces the overhead (e.g. by 50% for the 3D approximating wavelets). This is interesting, since it extends beyond linear B-spline wavelets [19]. That is, all tensor product wavelets have a rectilinear stencil, the boundary of which always has zero-valued coefficients. The multilinear optimization ('Min multilinear') gives another significant reduction, and the use of the radial interpolant ('Min radial') reduces it by half again, with respect to the original rectilinear block.

More importantly, these savings translate to runtime representations in practice (see lower left chart in Figure 9). The normalized multilinear and radial meshes induce an overhead of about 2–5 vertices per wavelet for unsaturated (Type I) and 1–2 vertices per wavelet when the mesh wavelets are saturated. Compared to the full stencil, the radial representation requires about half to a quarter of the vertices for the normalized wavelets, and about half of the total number of supercubes (our encoding primitive). On the other hand, the increase in stencil size due to our conservative radial interpolant stencils $(r, slice)$ compared to the optimal (r, opt) incurs a slight overhead (roughly 5–10%).

Figure 6 illustrates the performance benefits of our adaptive representation as a function of per-cell time associated with a hypothetical processing task such as isocontouring or volume rendering. Rather than arbitrarily choosing such a task, we parameterize the speedup by the processing time. Although the per-cell time needed just to extract our adaptive meshes is around 8x longer than the per-cell time needed to traverse the full-resolution grid, our method greatly reduces the number of cells and therefore the total processing time. The top curve in this figure shows the breakeven point, i.e. the highest fraction of adaptive grid cells to full-resolution grid cells, for which our adaptive method yields the same total processing time as working with the full-resolution grid. Because our adaptive meshes usually retain 1% or fewer cells, we tend to achieve speedups of 10x or more.

6.4 Thresholding by function domain and range

For isosurfaces and interval volumes, we often require the field to be accurate only within a range of function values. We here evaluate the efficiency of representing only a scaffold of vertices around a given isocontour from a signed distance field generated from a triangle mesh. Specifically, we discard those wavelets whose support does not intersect the zero set, allowing us to adapt surface detail by thresholding the remaining wavelets by magnitude (similar to [17, 24, 48]).

We observe that the distance field has sharp singularities related to the 'medial' regions – locations that are equally distant from several points on the surface. By filtering with respect to a range of isovalues, we are able to remove these singularities that we are not actually interested in, while retaining all details necessary to perfectly reconstruct the surface using less than 1% of wavelet coefficients (see Figure 5).

We evaluate the quality of our approximation by computing the mean squared distance, as given by our approximate field, evaluated over the known surface. We relate this error to the error of the full-resolution (sampled) distance field, which is non-zero since the distance field is generally not trilinear. Figure 5(right) demonstrates the quality benefit of using isosurface thresholded wavelets ('Iso' suffix). Specifically, the non-filtered distance field requires an order of magnitude more vertices to achieve the same error tolerance.

It is also common to select wavelets based on a subset of the function's domain. In Figure 7 we demonstrate a region of interest query, illustrating the gradual decrease in resolution outside the circular query region, within which the function is shown at full detail.

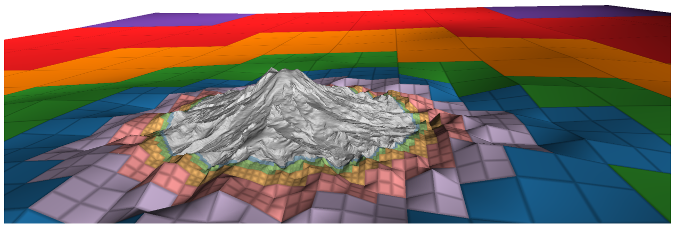


Fig. 7. Region of interest filtering on the 4097^2 Puget Sound dataset. The ROI is a circle with radius 400 samples centered around Mt. Rainier. The mesh contains 130 K vertices in 11.4 K supercubes (average occupancy 97%), comprising less than 1% of the total field. Note how the bilinear patches (colored by quadtree depth) define a globally C^0 function.

6.5 Effects of the interpolant: linear vs. multilinear

Since adaptive octrees generate non-conforming meshes (i.e. they have hanging nodes), the standard approach when approximating a function adaptively over an octree is to triangulate the cubes into simplices, either using red-green patterns [20, 33] or bisection-based triangulations [47]. The field is then approximated using a linear interpolant.

Here, we demonstrate the benefits of using the interpolant associated with the underlying wavelet. For a given adaptive octree mesh with reconstructed function values at the cube corners, we compare reconstructing each cube using either a multilinear interpolant or a linear interpolant based on simplex bisection [47]. An analysis of the peak signal to noise ratio (PSNR) suggests that multilinear interpolation can yield significant improvements in fidelity for both natural images and volumetric data. For example, on the Rayleigh-Taylor instability (RTI) dataset [7] (Figure 9), we found that adaptive meshes defined by the same set of vertices can yield a PSNR improvement of several dB when using the native multilinear interpolant over the mesh cells rather than a linear interpolant over its tetrahedralization. See also the supplemental material for detailed comparisons.

6.6 Multilinear interpolation on subsampled meshes

The relationship between radial predictors and linear B-spline wavelets can also be beneficial when attempting to use a multilinear interpolant over a subsampled scalar field, i.e. where we skip the wavelet analysis phase and merely discard a subset of the samples that we do not care about. Subsampling has been successful for linear interpolants over triangulated meshes, where it forms the basis of numerous interactive LOD schemes e.g. for view-dependent terrain visualization [15] and view-dependent isosurface extraction [18], among numerous others.

Due to the centrality of multilinear interpolants in many visualization applications (e.g. the Marching Cubes algorithm [27] and direct volume rendering [1]), we would like to directly apply a multilinear interpolant to a subsampled regular grid. A natural choice would be to edge-balance the mesh (see Section 2), as in several adaptive isosurface extraction techniques [37, 50], and to apply a multilinear interpolant on the cubes of the mesh. Unfortunately, although this approach defines a globally C^0 function, it leads to disturbing visual artifacts (see Figure 8(a)). However, when we apply a radial interpolant to the same mesh, the artifacts disappear (see Figure 8(b)), and we have achieved a reconstruction equivalent to our interpolating linear B-spline wavelet approximating the original function.

To see why the radial interpolant helps, recall that interpolating wavelets have only the predictor stage (w-lift) but not the update stage (s-lift) in the lifting scheme. Thus, an interpolating wavelet's coefficient is associated with a (primal) face f of the mesh, and its interpolated value depends only on the values of the corner vertices of f. Thus, a subsampled mesh corresponds to an interpolating wavelet synthesis defined by all of the retained vertices in our subsampled mesh. By following our octree reconstruction algorithm (Section 5.2.1) we see that for the interpolating 'Min radial' extraction, we need only refine all octree parents of the vertices of the dual cubes of these vertices (giving an edge-balanced mesh), and radially interpolate the cells.

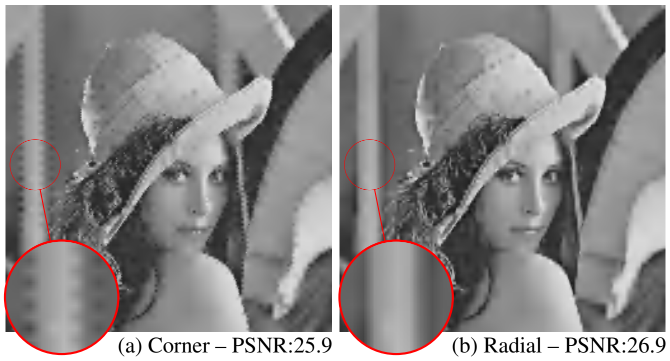


Fig. 8. (a) Artifacts when using a corner-based bilinear predictor to balance a subsampled adaptive mesh. (b) Radial spectral predictors based on corners and edges contain the complete support of their underlying basis function. Highlighted areas (red circles) are magnified by 2.25x.

7 Conclusions

In this paper, we have developed a framework based on adaptive octree meshes for efficiently encoding and reconstructing fields defined by tensor product wavelets. This framework enables us to represent simplified functions at runtime in an output sensitive manner. Applications like isosurfacing, DVR, ray tracing, and image panning can benefit from the reduction in both storage and computation when working with such an adaptive mesh. We have analyzed the hierarchical dependency relation among the wavelets to define an adaptive refinement strategy that allows extracting the coarsest mesh for representing any combination of wavelets. Using this dependency relation, we have demonstrated the potential benefits of saturating the mesh with the set of wavelets whose stencils are already present in the mesh. When evaluating the function on the interior of the cells of the mesh, we have demonstrated the quality benefits of using the interpolant induced by the wavelet scheme rather than a piecewise linear interpolant on a simplicial subdivision of the mesh, as in previous work [3, 20, 47].

In the case of linear B-spline wavelets, we have empirically and visually demonstrated quality improvements when using the wavelet scheme's underlying multilinear interpolant on the cubic cells of the octree rather than a linear interpolant on a simplicial subdivision.

We have demonstrated the benefits of the proposed approach using several different wavelet filtering strategies, including coefficient thresholding, region of interest queries, and isovalue-based thresholding. In most cases, we can achieve a faithful approximation to the desired features of the underlying function using only a small fraction of the space necessary for the original domain on which the problem was defined.

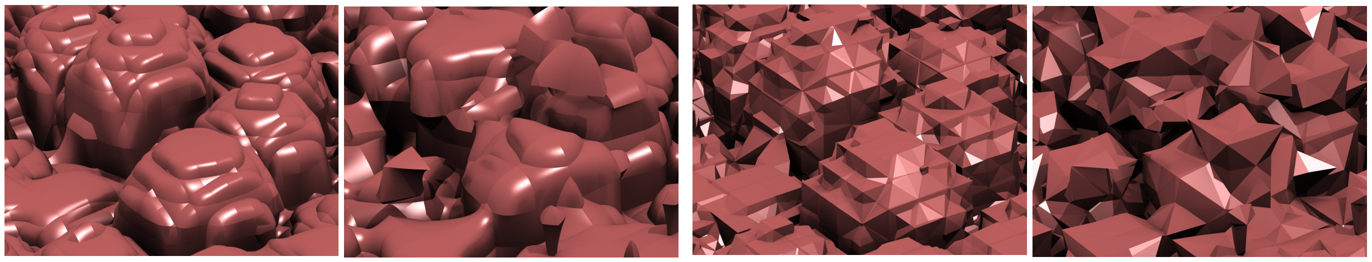
A limitation of the current work is that modification of the function (e.g. due to the insertion of a new wavelet coefficient) is not a local operation, since it could require cascades of updates over the entire support of the wavelet. In the worst case, insertion of a wavelet could require all vertices in the adaptive mesh to update, for example, if the wavelet associated with the root of the octree is added to a mesh. As part of our ongoing work, we are developing a refinement scheme that will allow incremental updates to the mesh.

As future work, we would like to implement adaptive wavelet transforms for other tensor product wavelets. An interesting generalization would be to cubic B-spline wavelets (with more than two lifting steps), as this would make our scheme globally C^1 continuous.

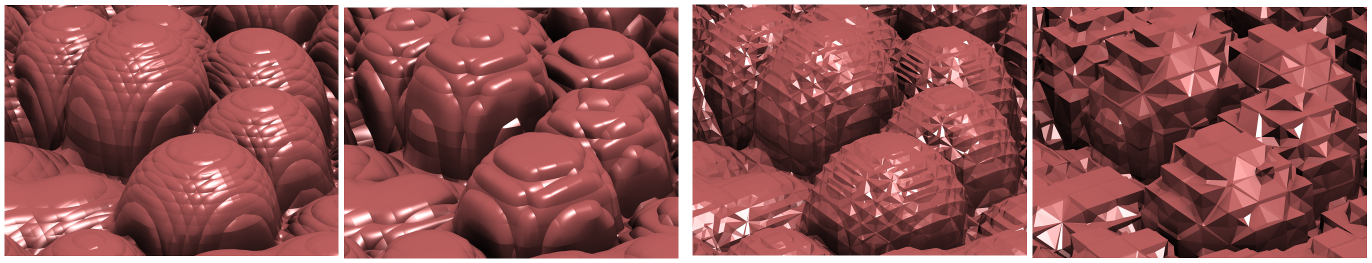
Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344, and was supported by the Office of Advanced Scientific Computing Research. We wish to thank the reviewers for their valuable feedback and the various data providers for making their data sets freely available.

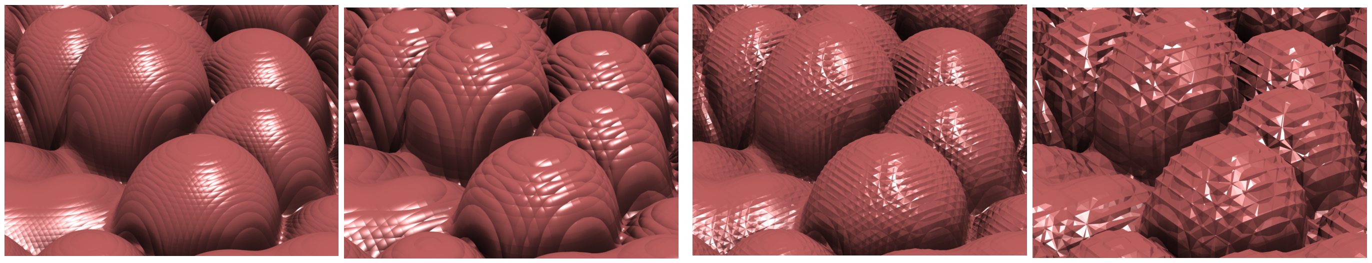
TRILINEARLY INTERPOLATED LINEARLY INTERPOLATED



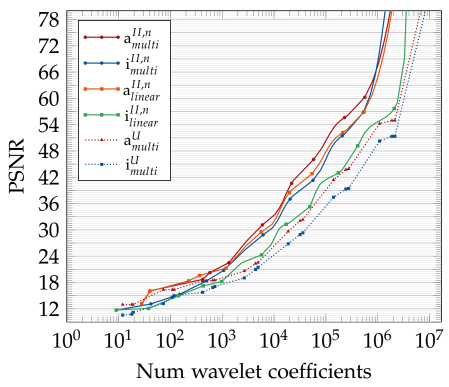
$a^{II,n}_{m,T:0.1\%}$ (23.8 K, 39.4 K, 1.7 K); PSNR: 41.3 $a^{U}_{m,D:13}$ (18.7 K, 35.9 K, 888); PSNR: 29.6 $a^{II,n}_{l,T:0.1\%}$ (21.9 K, 50.0K, 1.9 K); PSNR: 39.1 $a^{U}_{l,D:13}$ (18.7 K, 35.9 K, 888); PSNR: 28.4

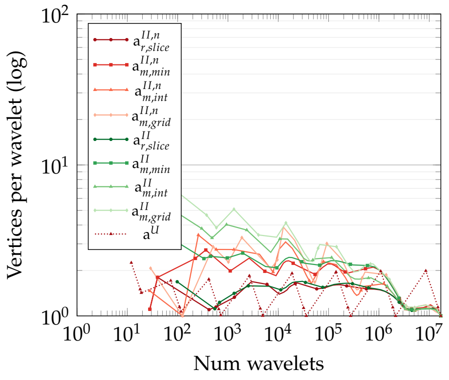


$a^{II,n}_{m,T:1\%}$ (265 K, 415 K, 17.4 K); PSNR: 56.2 $a^{U}_{m,D:17}$ (242 K, 275 K, 5.8 K); PSNR: 43.7 $a^{II,n}_{l,T:1\%}$ (248 K, 506 K, 19.0 K); PSNR: 52.8 $a^{U}_{l,D:17}$ (242 K, 275 K, 5.8 K); PSNR: 40.6



$a^{II,n}_{m,T:10\%}$ (2.83 M, 3.36 M, 74.2 K); PSNR: 99.0 $a^{U}_{m,D:20}$ (1.89 M, 2.15 M, 41.7 K); PSNR: 55.0 $a^{II,n}_{l,T:10\%}$ (2.80 M, 3.49 M, 75.5 K); PSNR: 97.3 $a^{U}_{l,D:20}$ (1.89 M, 2.15 M, 41.7 K); PSNR: 51.5





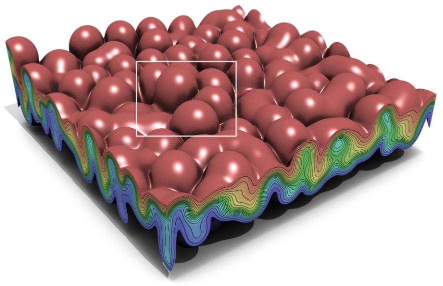


Fig. 9. Surfaces of ray traced interval volumes on the 257^3 Rayleigh-Taylor instability dataset [7] comparing approximating $a^{II,n}$ wavelets (raytraced at full resolution with tricubic B-spline interpolation on bottom right) for meshes defined by similar number of wavelet coefficients (rows). Columns show close-ups of adaptive meshes extracted using (normalized and saturated) wavelet magnitude thresholded meshes (columns 1 and 3) and progressive uniform meshes at a given depth in the hierarchy (columns 2 and 4) when interpolated trilinearly over the cubes of the adaptive octree (columns 1 and 2) and linearly over a corresponding tetrahedralized mesh (columns 3 and 4). Rows show: 0.1% thresholded wavelets (T: 0.1%) vs. uniform depth 13 (D: 13); 1% thresholded vs. depth 17 (T: 1% vs. D: 17) and 10% thresholded vs. depth 20 (T: 10% vs. D: 20). When thresholding above T: 10%, the images are virtually indistinguishable from those of the mesh at full resolution. The numbers below each figure indicate the number of wavelets, vertices and supercubes in the corresponding mesh as well as the PSNR of the approximation. Note that the surface artifacts are due to the lack of normal continuity of the interpolant across cell boundaries (trilinear on the left, linear on the right). When the interpolant is fixed (columns 1 vs. 2, and 3 vs. 4), wavelet filtering achieves similar PSNR using an order of magnitude fewer vertices. Alternatively, for a fixed mesh (columns 1 vs. 3 and 2 vs. 4), multilinear interpolants achieve a gain of several decibels PSNR on the same mesh. The chart on the lower left provides a broader overview of the experiment from which this example was selected and the chart on the lower middle indicates the efficiency of the representation for different stencils in terms of the number of vertices per wavelet using the saturated (Type II, middle) wavelets.

References

[1] M. Ament, D. Weiskopf, and H. Carr. Direct interval volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1505–1514, 2010.

[2] L. Balmelli. *Rate-distortion optimal mesh simplification for communications*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne EPFL, 2000.

[3] L. Balmelli, T. Liebling, and M. Vetterli. Computational analysis of mesh simplification using global error. *Computational Geometry Theory and Applications*, 25(3):171–196, 2003.

[4] R. Bank, A. H. Sherman, and A. Weiser. Refinement algorithms and data structures for regular local mesh refinement. In R. Stepleman, M. Carver, R. Peskin, W. F. Ames, and R. Vichnevetsky, editors, *Scientific Computing, IMACS*, volume 1, pages 3–17. North-Holland, Amsterdam, 1983.

[5] M. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82(1):64–84, 1989.

[6] M. Bertram, M. Duchaineau, B. Hamann, and K. Joy. Generalized B-spline subdivision-surface wavelets for geometry compression. *IEEE Transactions on Visualization and Computer Graphics*, 10(3):326–338, May-June 2004.

[7] W. Cabot and A. Cook. Reynolds number effects on Rayleigh-Taylor instability with possible implications for type Ia supernovae. *Nature Physics*, 2(8):562–568, 2006.

[8] H. Carr, T. Moller, and J. Snoeyink. Artifacts caused by simplicial subdivision. *IEEE Transactions on Visualization and Computer Graphics*, 12(2):231–242, 2006.

[9] N. Chegini and R. Stevenson. The adaptive tensor product wavelet scheme: Sparse matrices and the application to singularly perturbed problems. *IMA Journal of Numerical Analysis*, 32(1):75–104, 2011.

[10] P. Cignoni, F. Ganovelli, C. Montani, and R. Scopigno. Reconstruction of topologically correct and adaptive trilinear isosurfaces. *Computers & Graphics*, 24(3):399–418, 2000.

[11] R. Claypoole, Jr., R. Baraniuk, and R. Nowak. Adaptive wavelet transforms via lifting. In *Proceedings IEEE Acoustics, Speech and Signal Processing*, volume 3, pages 1513–1516, May 1998.

[12] A. Cohen, I. Daubechies, and J. Feauveau. Biorthogonal bases of compactly supported wavelets. *Communications on pure and applied mathematics*, 45(5):485–560, 1992.

[13] I. Daubechies and W. Sweldens. Factoring wavelet transforms into lifting steps. *Journal of Fourier analysis and applications*, 4(3):247–269, 1998.

[14] M. Domingues, S. Gomes, O. Roussel, and K. Schneider. Adaptive multiresolution methods. *Proceedinges ESAIM*, 32:1–96, 2011. Summer school on multiresolution and Adaptive Mesh Refinement methods.

[15] M. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein. ROAMing terrain: Real-time Optimally Adapting Meshes. In *Proceedings IEEE Visualization*, pages 81–88, 1997.

[16] H. Edelsbrunner and M. Kerber. Dual complexes of cubical subdivisions of R^n. *Discrete & Computational Geometry*, 47(2):393–414, 2012.

[17] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings SIGGRAPH*, pages 249–254, New Orleans, LA, July 2000. ACM Press.

[18] B. Gregorski, M. Duchaineau, P. Lindstrom, V. Pascucci, and K. Joy. Interactive view-dependent rendering of large isosurfaces. In *Proceedings IEEE Visualization*, pages 475–484. IEEE Computer Society Washington, DC, USA, October 2002.

[19] E. Grinspun, P. Krysl, and P. Schröder. CHARMS: A simple framework for adaptive simulation. *ACM Transactions on Graphics*, 21(3):281–290, July 2002.

[20] M. Gross, O. Staadt, and R. Gatti. Efficient triangular surface approximations using wavelets and quadtree data structures. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):130–143, June 1996.

[21] P. Heckbert. Survey of texture mapping. *IEEE Computer Graphics & Applications*, 6(11):56–67, 1986.

[22] M. Holmström. Solving hyperbolic PDEs using interpolating wavelets. *SIAM Journal on Scientific Computing*, 21(2):405–420, 1999.

[23] L. Ibarria, P. Lindstrom, and J. Rossignac. Spectral interpolation on 3 x 3 stencils for prediction and compression. *Journal of Computers*, 2(8):53–63, 2007.

[24] D. Laney, M. Bertram, M. Duchaineau, and N. Max. Multiresolution distance volumes for progressive surface compression. In *Proceedings 3D Data Processing Visualization and Transmission*, pages 470–479, 2002.

[25] L. Linsen, B. Hamann, and K. Joy. Wavelets for adaptively refined "3rd-root-of-2" subdivision meshes. *International Journal of Computers & Applications*, 29(3):223–231, 2007.

[26] L. Linsen, V. Pascucci, M. Duchaineau, B. Hamann, and K. Joy. Wavelet-based multiresolution with $\sqrt[n]{2}$ subdivision. *Journal on Computing, Special Edition: Dagstuhl Seminar on Geometric Modelling*, 72(1–2):129–142, 2004.

[27] W. Lorensen and H. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *Proceedings SIGGRAPH*, pages 163–169. ACM Press New York, NY, USA, 1987.

[28] S. Mallat. *A wavelet tour of signal processing*. Academic press, 2008.

[29] P. Moran and D. Ellsworth. Visualization of AMR data with multi-level dual-mesh interpolation. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1862–1871, 2011.

[30] H. Müller and M. Stark. Adaptive generation of surfaces in volume data. *The Visual Computer*, 9(4):182–199, 1993.

[31] G. M. Nielson. On marching cubes. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):283–297, 2003.

[32] B. Payne and A. Toga. Surface mapping brain function on 3D models. *IEEE Computer Graphics & Applications*, 10(5):33–41, Sept. 1990.

[33] S. Plantinga and G. Vegter. Isotopic meshing of implicit surfaces. *The Visual Computer*, 23(1):45–58, 2007.

[34] A. Said and W. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(3):243–250, June 1996.

[35] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. The Morgan Kaufmann series in computer graphics and geometric modeling. Morgan Kaufmann, 2006.

[36] J. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing*, 41(12):3445–3462, 1993.

[37] R. Shekhar, E. Fayyad, R. Yagel, and J. Cornhill. Octree-based decimation of marching cubes surfaces. In *Proceedings IEEE Visualization*, pages 335–342, Los Alamitos, CA, USA, 1996. IEEE Computer Society.

[38] R. Shu, C. Zhou, and M. Kankanhalli. Adaptive marching cubes. *The Visual Computer*, 11(4):202–217, 1995.

[39] J. Stam. Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values. In *Proceedings ACM SIGGRAPH*, pages 395–404, New York, NY, USA, 1998. ACM.

[40] E. Stollnitz, A. DeRose, and D. Salesin. *Wavelets for Computer Graphics: Theory and Applications*. Morgan Kaufmann, 1996.

[41] W. Sweldens. The lifting scheme: A construction of second generation wavelets. *SIAM Journal on Mathematical Analysis*, 29(2):511–546, 1998.

[42] The volume library. http://www9.informatik.uni-erlangen.de/External/vollib.

[43] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, April 2004.

[44] G. Weber, O. Kreylos, T. Ligocki, J. Shalf, H. Hagen, B. Hamann, and K. Joy. Extraction of crack-free isosurfaces from adaptive mesh refinement data. In G. Farin, B. Hamann, and H. Hagen, editors, *Hierarchical and Geometrical Methods in Scientific Visualization*, Mathematics and Visualization, pages 19–40. Springer Berlin Heidelberg, 2003.

[45] K. Weiss and L. De Floriani. Diamond hierarchies of arbitrary dimension. *Computer Graphics Forum*, 28(5):1289–1300, 2009.

[46] K. Weiss and L. De Floriani. Supercubes: A high-level primitive for diamond hierarchies. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1603–1610, November-December 2009.

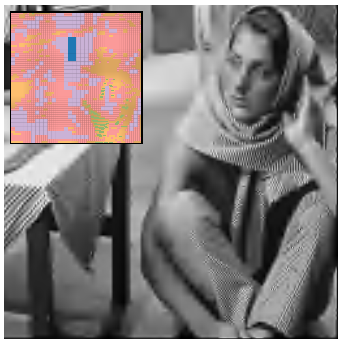
[47] K. Weiss and L. De Floriani. Bisection-based triangulations of nested hypercubic meshes. In S. Shontz, editor, *Proceedings 19th International Meshing Roundtable*, pages 315–333, October 2010.

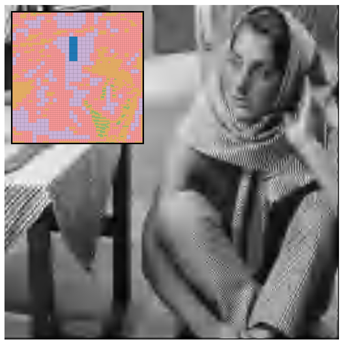
[48] K. Weiss and L. De Floriani. Isodiamond hierarchies: An efficient multiresolution representation for isosurfaces and interval volumes. *IEEE Transactions on Visualization and Computer Graphics*, 16(4):583 – 598, July-Aug. 2010.

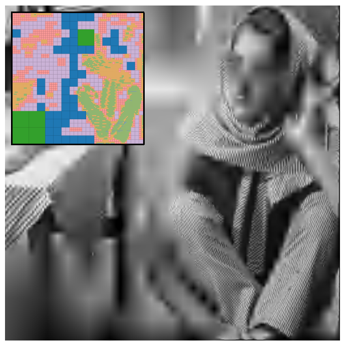
[49] K. Weiss and L. De Floriani. Simplex and diamond hierarchies: Models and applications. *Computer Graphics Forum*, 30(8):2127–2155, 2011.

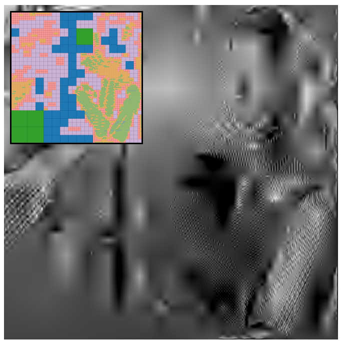
[50] R. Westermann, L. Kobbelt, and T. Ertl. Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. *The Visual Computer*, 15(2):100–111, 1999.

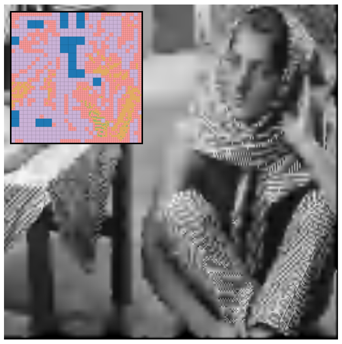
[51] Y. Zhang and C. Bajaj. Adaptive and quality quadrilateral/hexahedral meshing from volumetric data. *Computer Methods in Applied Mechanics and Engineering*, 195(9-12):942–960, 2006.

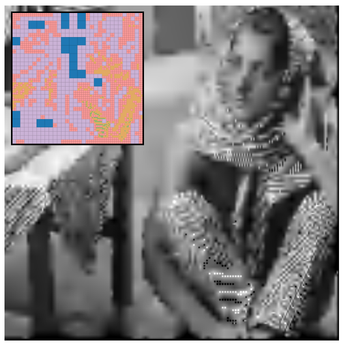

$a^{II,n}$ (6.3 K, 7.4 K, 1.3 K); PSNR: 23.2; SSIM .76

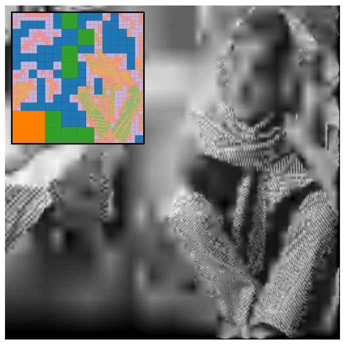

$a^{I,n}$ (2.6 K, 7.4 K, 1.3 K); PSNR: 22.9; SSIM .74

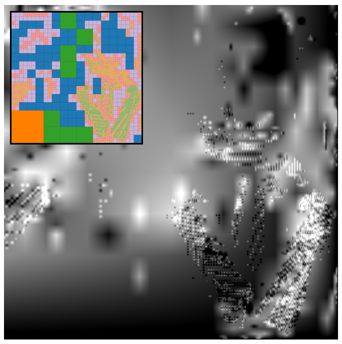

a^{II} (11.7 K, 13.8 K, 2.5 K); PSNR: 21.8; SSIM .70


a^{I} (2.6 K, 13.8 K, 2.5 K); PSNR: 15.1; SSIM .47


$i^{II,n}$ (5.0 K, 5.2 K, 965); PSNR:21.0; SSIM:.66


$i^{I,n}$ (2.6 K, 5.2 K, 965 K); PSNR:20.4; SSIM:.63


i^{II} (9.5 K, 10.0 K, 1.9 K); PSNR:18.3; SSIM:.60


i^{I} (2.6 K, 10.0 K, 1.9 K); PSNR:13.0; SSIM:.36

Comparison of the effects of saturation and normalization on the approximating (top row) and interpolating (bottom row) 513^2 Barbara dataset thresholded at 1% of wavelet coefficients ($\approx 2.6\ K$ coefficients). Columns show (left to right): normalized saturated ($\cdot^{II,n}$), normalized unsaturated ($\cdot^{I,n}$), unnormalized saturated ($\cdot^{II}$) and unnormalized unsaturated ($\cdot^{I}$) approximations. We report the number of wavelet coefficients $|\psi|$, vertices $|v|$, and supercubes $|s|$, in the reconstructed meshes (listed as ($|\psi|$, $|v|$, $|s|$)) as well as their Peak Signal to Noise Ratio (PSNR) and structural similarity (SSIM) [43]. Inlays show the cells of the corresponding quadtrees, colored by hierarchy level. We observe that the saturated variants (columns 1 and 3) are able to achieve higher fidelity (in terms of PSNR and SSIM) than their unsaturated counterparts (columns 2 and 4) over the same domain decomposition. We also observe that the unnormalized variants (columns 3 and 4) spend a significant portion of their wavelet budget trying to resolve high resolution features of the underlying function, while the normalized variants (columns 1 and 2) select important wavelets from coarser levels of resolution. Thus the normalized meshes can reconstruct higher fidelity meshes using fewer wavelet coefficients. Finally, we observe that the approximating wavelets (top row) achieve better reconstruction fidelity than interpolating wavelets (bottom row) defined by the same number of wavelet coefficients, but require a larger support mesh.