This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2021.3074607, IEEE Transactions on Visualization and Computer Graphics



# Interactive Focus+Context Rendering for Hexahedral Mesh Inspection

Christoph Neuhauser, Junpeng Wang, and Rüdiger Westermann

**Abstract**—The visual inspection of a hexahedral mesh with respect to element quality is difficult due to clutter and occlusions that are produced when rendering all element faces or their edges simultaneously. Current approaches overcome this problem by using focus on specific elements that are then rendered opaque, and carving away all elements occluding their view. In this work, we make use of advanced GPU shader functionality to generate a focus+context rendering that highlights the elements in a selected region and simultaneously conveys the global mesh structure and deformation field. To achieve this, we propose a gradual transition from edge-based focus rendering to volumetric context rendering, by combining fragment shader-based edge and face rendering with per-pixel fragment lists. A fragment shader smoothly transitions between wireframe and face-based rendering, including focus-dependent rendering style and depth-dependent edge thickness and halos, and per-pixel fragment lists are used to blend fragments in correct visibility order. To maintain the global mesh structure in the context regions, we propose a new method to construct a sheet-based level-of-detail hierarchy and smoothly blend it with volumetric information. The user guides the exploration process by moving a lens-like hotspot. Since all operations are performed on the GPU, interactive frame rates are achieved even for large meshes.

**Index Terms**—Visualization of Hex-Meshes, Real-Time Rendering, GPUs.

## 1 Introduction

Hexahedral elements have widespread use in numerical simulation methods using finite elements and finite volumes. Therefore, hexahedral mesh generation (hex-meshing) has become a topic of intense research. However, for all but simple volumetric bodies it is impossible to construct a deformation-free hexahedral mesh, i.e., where the elements are rectilinear cubes (cuboids), that accurately represents the boundary of the body or aligns with specific material features in the interior. Thus, it is one of the grand challenges in hex-meshing to construct meshes with as low as possible deformations [2], [3], [4].

Visual mesh inspection tools support experts in assessing the specific deformation characteristics of the output produced by different meshing techniques. The following mesh analysis tasks are typically performed:

- T1 Assessment of the spatial distribution of the cell deformations, to reveal whether a hex-mesh produces clusters of cells with large deformations or uniformly distributes medium deformations across the domain.
- T2 Fine-granular inspection of a selected group of cells, to assess similarities and variations in their geometric deformations.
- T3 Investigation of the relationships between occurring deformations and specific geometric features of the meshed body, to understand the interplay between object boundaries and the internal mesh structure.
- T4 Assessment of the relationships between simulation results or accuracy and cell deformations, to shed light on the effects of different meshing techniques on the simulation output.

• *All authors are with the Computer Graphics & Visualization Group, Technische Universität München, Garching, Germany.*
*E-mail: {christoph.neuhauser, junpeng.wang, westermann}@tum.de.*

A visual inspection is difficult, however, since often high deformations occur in the interior of the volumetric body. In principle, direct volume rendering techniques for unstructured volumetric meshes can be used to render the 3D deformation field (Fig. 2 (left)). This can effectively communicate the spatial distribution of deformations (T1), yet the cell structure is entirely lost. Since drawing all edges results in extreme clutter and occlusions, visualization tools for hex-meshes often restrict the analysis to the boundary faces of the 3D mesh (Fig. 2 (middle)), and provide options to clip subsets of elements so that interior faces appear (Fig. 2 (right)).

Bracci et al. [5] provides means to peel away layers of elements from outside to inside, remove cells below a certain deformation value, and reveal hidden irregular edges via transparency rendering. This enables an interactive fine-granular visual exploration of hex-meshes (T2), but since there is no guidance to the important regions at first hand, the user needs to actively search through the mesh by successive cell filtering operations. Especially when low deformation cells are surrounded by high deformation cells, this procedure can become misleading. Furthermore, successive filtering makes it difficult to maintain the global mesh structure and spatial relationships between mesh regions with different properties. Xu and Chen [6] address this problem by showing a global topological mesh structure in addition to selected cells (T3). The spatial distribution of deformations, however, can only be accessed via animation, i.e., by stepping through the sets of cells in deformation space, which is problematic because of the required memorization capabilities and attention shifts [7].

### 1.1 Contribution

In this work, we extend current visualization techniques for hex-meshes by a combination of face-based volume rendering with fragment-based edge rendering. Our goal is to enable users to focus their visualization efforts on the area of their choice, while preserving a surrounding context that conveys important positional

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2021.3074607, IEEE Transactions on Visualization and Computer Graphics



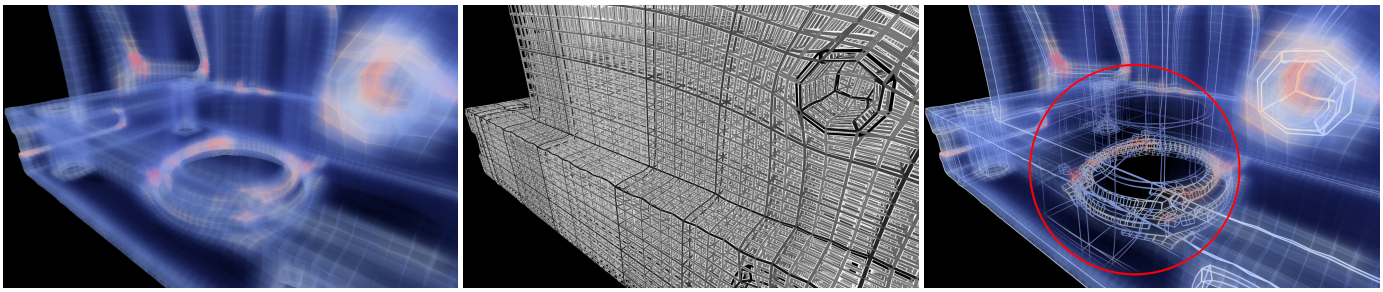

Fig. 1. From left to right: Contextual visualization using face-based volume rendering, fragment-based edge rendering, our proposed focus+context (F+C) rendering using smooth blending between edge and volumetric rendering including contextual lines. Model courtesy of [1].

cues and hints toward non-explored regions of interest. In the global context view the deformation field is rendered as a semi-transparent volumetric field in combination with few contextual edges (T1), so that important regions and the spatial mesh structure can quickly be recognized (T3). This enables in particular to simultaneously visualize the mesh structure and a volumetric field given as scalar values at the mesh vertices or cells (T4). The detailed focus view is selected via a user-defined screen space lens with depth focus, in which edge-based rendering is used (T2).

One of our goals is to obtain a smooth transition from edge-based focus rendering to volumetric context rendering, which is important to allow the user to easily infer the relationship between the focus and context structures. Therefore, we introduce a GPU renderer for hex-meshes that solely renders cell faces and performs all operations that change the mesh appearance in a fragment shader. The shader smoothly blends between different rendering options depending on where a fragment is located w.r.t. the focus region and whether a fragment is an edge or interior face fragment. Furthermore, we incorporate an edge-based level-of-detail (LoD) structure into the renderer, to adapt the density of rendered mesh edges depending on cell deformation and distance to the focus center. The coarser LoDs further serve as shape cues in the context region. Since all rendering options are performed solely on the fragment level, a smooth blending from sharp details to a more fuzzy appearance with embedded characteristic edges as shape cues can be performed efficiently. Our proposed visualization technique builds upon the following specific contributions:

- A single-pass GPU renderer with fragment shader-based edge and volume rendering including transparency.
- A LoD line hierarchy that is extracted from a hex-mesh using a topological subdivision scheme based on mesh sheet elements.
- A rendering technique that smoothly blends between focus and context, by continuously adapting edge density as well as edge and face appearance.

In Sec. 2 and Sec. 3, we first discuss related work and provide an overview of our technique. In Sec. 4, we discuss how a smooth transition between focus and context is achieved via edge- and face-specific modulations in a pixel shader. Sec. 5 is dedicated to the extraction of a hierarchical LoD, which is required to convey important positional cues. Finally, we perform a detailed quality and performance analysis that justifies the feasibility of our approach even for meshes comprising millions of elements. The code is published on https://github.com/chrismile/HexVolumeRenderer.

## 2 Related Work

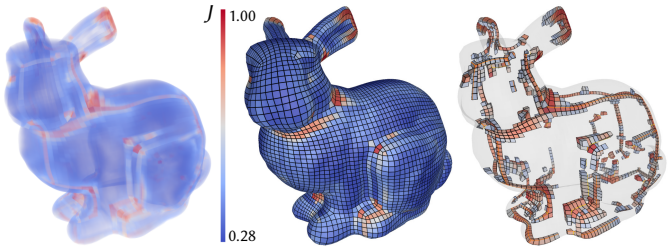


Fig. 2. Conventional hex-mesh visualizations. $J$ is the Jacobian ratio of each cell. From left to right: Direct volume rendering, the boundary surface, filtering cells with low deformation. Model courtesy of [3].

Visualization techniques for hexahedral grids can be divided into surface-based and direct volume rendering techniques. Surface-based techniques render hexahedral elements as opaque cuboids, including wireframe rendering and face coloring to emphasize certain element properties. For a thorough overview of the different rendering styles that are used in modelling applications, let us refer to the recent work by [5]. They also introduced novel line-based visualization options to maintain the overall model structure and emphasize singular edges in a hex-mesh. For computing the deformation of cells, our implementation uses the code from [5], which implements various measures for cell deformation supported by the Verdict library [8]. A summary and discussion of different quality metrics for hex-meshes is given by [9]. The code from [2] is used for loading and processing hexahedral meshes.

Recently, [6] proposed to visualize the mesh structure of hexahedral meshes by using a subset of the most important base-complex sheets and dual chords, and show their interrelation using adjacency matrices. We take inspiration from their approach utilizing base-complex mesh sheets to reduce the structural complexity of a mesh (cf. Sec. 5). Our approach uses hexahedral sheets [10], [11] instead of base-complex sheets, and merges sheets for creating a LoD structure instead of directly visualizing a subset of them. The use of hexahedral sheets for hex-mesh construction, simplification and reparameterization is thoroughly discussed in the work by [12].

Direct volume rendering of hexahedral meshes has a long tradition in volume visualization, and many of the concepts that are used by more recent works are discussed in the survey by [13]. Our GPU-based approach shares similarities with cell projection techniques w.r.t. how the cells are rendered and their visibility order is established. Cell projection techniques exploit the GPU to efficiently render triangles and perform linear interpolation of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2021.3074607, IEEE Transactions on Visualization and Computer Graphics



per-vertex attributes for each rendered fragment. Cuboids are first decomposed into tetrahedra, and then rasterized and blended using the GPU [14], [15], [16], [17]. [15] utilized the GPU for visibility sorting of rendered fragments, which is conceptually similar to the approach we employ for visibility sorting using per-pixel fragment lists [18], a GPU realization of the A-buffer [19] to store the unordered set of fragments falling into each pixel. These fragments are then sorted explicitly based on the stored depth information. Recently, SparseLeap [20] has been introduced as a pyramidal occupancy map to generate geometric structures representing non-empty regions, which makes use of per-pixel fragment lists to determine occupied space and accelerate volume ray-casting.

Related to our visualization technique is the use of transparency and adaptive primitive density for streamline rendering. When too many lines are shown simultaneously, occlusions and visual clutter are quickly introduced. While we address this by smoothly blending into a volumetric context and using few representative edge sequences from coarser LoDs, others have proposed importance- and similarity-based criteria in screen-space to select the rendered lines dynamically on a frame-to-frame basis. Screen-space approaches determine for each new view the subset of lines to be rendered so that occlusions are reduced and more important lines are favored over less important ones [21], [22], [23]. The amount of occlusion is determined by the "overdraw", i.e., the number of projected line points [21], [23] or the maximum projected entropy [22] per pixel. [24] decrease the opacity of non-important foreground lines using per-frame opacity optimization. A summary and evaluation of different GPU transparency rendering techniques for large line sets is given by [25]. [26] build a line hierarchy to continuously decrease the density of less important lines.

## 3 Method Overview

Our method renders all hex-faces as a quadrilateral formed by two triangles. A fragment shader determines the appearance of each fragment depending on whether it lies in the focus or context region, and further depending on whether it is an "edge fragment", i.e., lying closer to a face edge than a given edge thickness, or a "face fragment", i.e., lying too far away from any of the face edges. This classifies each fragment into 4 different types that determine how it is shaded (see Fig. 3). While in the context region a more volumetric appearance with subtle edge accentuation is used, in the focus region only the edges are clearly emphasized. Edge and face colors and opacities are made dependent on the importance measure, i.e., the strength of cell deformation, so that also in the context region important cells are emphasized. We use the Jacobian ratio metric as deformation measure [9], which is deduced from the Jacobian matrix that is related to the transformation of a cuboid into the deformed cell. Since the importance measure is cell-based, every vertex gets assigned the maximum importance value of all cells sharing this vertex. The triangle rasterizer then brings the interpolated importance values to the fragments. In addition, the maximum importance value of all cells sharing an edge are made available in the fragment shader for that edge.

The renderer changes the appearance smoothly from edge-based to volumetric with increasing distance to the focus center in screen space, as described in Sec. 4. Therefore, the opacity and width of edges in focus is smoothly decreased towards the focus border, and the color is blended towards the face colors used for rendering the context region. For each pixel, all fragments falling into that pixel are stored in a per-pixel fragment list on the GPU,

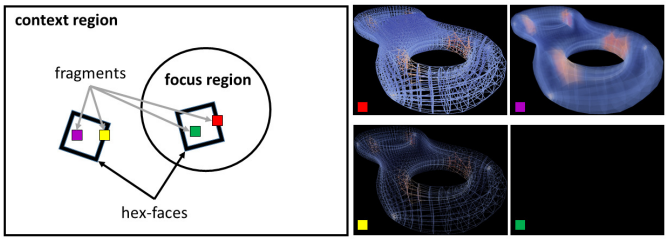


Fig. 3. Left: Classification of fragments depending on whether they are in focus or context, and whether they are close to a face edge or not. Right: Depending on the classification, the fragments take on different appearances. For each fragment, the image shows how the rendering looks like if only fragments of this type are rendered.

and they are sorted w.r.t. increasing distance to the camera. This enables opacity-based blending, i.e., $\alpha$-blending, of fragments in the correct visibility order. For sorting, we use a GPU-friendly implementation of priority queues [25].

The described rendering approach has two limitations: Firstly, in the focus region there can be many non-important edges that occlude important ones. Secondly, in the context region the basic mesh structure gets lost due to increasing volumetric appearance. To address these limitations, we construct a LoD line structure (Sec. 5), in which mesh edges are continually removed at coarser hierarchy levels. Fig. 4 illustrates how the LoD structure is used, by assigning to every edge the maximum level at which this edge is still present in the LoD structure. In the focus region, instead of removing edges with an importance value below a selected threshold, these edges are rendered if they are also present at some coarse LoD. We call these edges *contextual edges*. In the context region, only contextual edges are rendered to provide an overview of the shape of the hex-mesh.

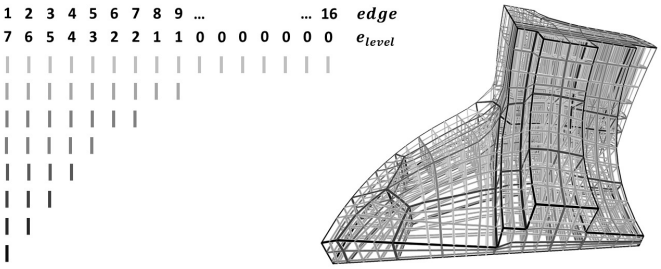


Fig. 4. Left: Edges are continually thinned out from level to level in the LoD hierarchy. Single edges get assigned the level at which they are last contained. Right: The LoD edge structure for a given hex-mesh. Greyscales from bright to dark encode LoD levels from fine to coarse. Model fandisk courtesy of [27].

## 4 Focus+Context

In the following, we describe how focus and context rendering is performed, and in particular how a smooth transition between both is achieved. A detailed discussion of the reference GPU implementation is given in Sec. 6. The user defines the focus region by positioning a circular lens with a *center* and controlled *radius* in screen space. The *focus* is 1 at the lens center and goes smoothly down to 0 towards its boundary.

Regardless of whether a fragment is finally shaded to appear as part of an edge or a face, hex-faces are rasterized with two triangles, and per-vertex attributes like the cell importance are barycentrically

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2021.3074607, IEEE Transactions on Visualization and Computer Graphics



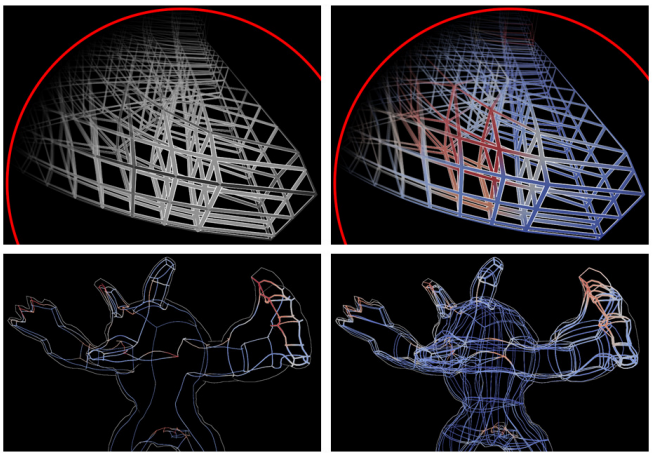

Fig. 5. Top: Focus edges are smoothly faded out with increasing distance to the focus center. Left: Edges colored by LoD level. Right: Edges colored by interpolated per-vertex deformation measure. Model fandisk courtesy of [27]. Bottom: Decreasing $lod$ from 11 (left) to 8 (right) increases the density of contextual edges. No focus selected. Model armadillo courtesy of [29].

interpolated. For every fragment, a fragment shader determines whether it should appear as an edge or a face. This is performed by first computing a fragment's screen space coordinate and its distance $dist$ to the focus center (normalized to range from 0 at the focus center to 1 at the focus boundary), and evaluating the $focus$ as $1 - smoothstep(0.7, 1, dist)$.

Then, a fragment's edge opacity $\alpha_e$, which determines whether the fragment belongs to a face ($\alpha_e = 0$) or an edge ($\alpha_e > 0$), is determined as follows:

$$\begin{aligned} w &= (1 + 0.3 \cdot focus) \cdot w_{base} \\ e_{focus} &= (d_{edge} \leq w \wedge (e_{level} \geq lod \vee e_{attr} \geq \delta)) \; ? \; 1 : 0 \\ e_{context} &= (d_{edge} \leq w \wedge e_{level} \geq lod) \; ? \; 1 : 0 \\ \alpha_e &= lerp(e_{context}, e_{focus}, focus) \end{aligned} \tag{1}$$

Here, $w_{base}$ is a minimum edge width, $d_{edge}$ is the fragment's shortest distance to any of the face edges in screen space, $e_{level}$ is the LoD level of the edge (Fig. 4), and $e_{attr}$ is the edge importance. First, the edge width is decreased with increasing distance to the focus center. Then, via $e_{context}$ and $e_{focus}$, respectively, it is determined whether the fragment belongs to an edge that should be rendered when lying in the focus or context region. In focus, an important edge is always rendered, i.e., $e_{attr}$ is greater than a selected importance threshold $\delta$. An unimportant edge is rendered only if it is at a user-selected coarse LoD level $lod$. In context, every edge with $e_{level} \geq lod$ is rendered. The final distance-based linear interpolation between $e_{context}$ and $e_{focus}$ transitions smoothly from focus to contextual edges.

The shader renders the focus edges with a thin white depth-dependent halo [28]. The halo gets thinner with increasing distance to the focus, and the edge colors are slightly darkened to make the edges stand out against the halo (Fig. 5 (top)). Focus edges blend into contextual edges, which are rendered as simple lines without a halo and colored according to the deformation measure. To maintain certain contextual edges as spatial cues in the focus and context region, the user can interactively select the value of $lod$ (Fig. 5 (bottom)).

Furthermore, fragments in the context that are in close vicinity to an edge, but are not visible at the selected LoD level, are slightly accentuated. If such a fragment doesn't belong to an edge according to Eqn. 1, $s_e$ determines how strongly it is emphasized:

$$s_e = d_{edge} \leq \frac{2}{3} \cdot w_{base} \; ? \; s : 1 \tag{2}$$

$s_e$ is used to enhance the face opacity $\alpha_f$ (see Egn. 4). Since $\alpha_f$ depends on the distance to the focus center (Eqn. 3), accentuated edges fade out accordingly. Fig. 6 demonstrates varying accentuation of contextual edges by variation of the accentuation strength $s$.

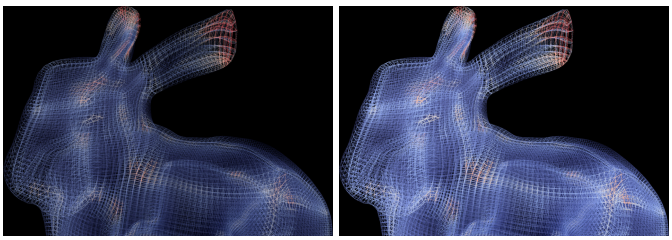

Fig. 6. Weakly ($s = 1.5$) and strongly ($s = 3$) accentuated edges according to Eqn. 2. No focus selected. Model bunny courtesy of [1].

Both parameters $\alpha_e$ and $s_e$ are used to assign the fragment opacity that emphasizes certain edges and smoothly blends between focus edges and contextual edges with increasing distance to the focus center. The edge colors are set via a color table that maps the edge importance values to colors $C_e$ (see Sec. 4.2). Since unshaded lines impact the ability to correctly observe spatial relations when looking at a still image (cf. Fig. 22), slightly desaturated fragments are drawn further away from the camera in the focus region. The user can further activate a circular lens boundary to indicate the currently selected focus point and area.

### 4.1 Contextual Face-Based Volume Rendering

If a fragment is not classified as part of an edge, it is rendered as part of a face to generate a volumetric appearance that hints to important mesh regions. In principle, once the face fragments are rendered and sorted in a fragment list, direct volume rendering using $\alpha$-compositing of cell contributions can be used (Fig. 7 (left)). This gives a continuous volumetric appearance, as if the object is filled with a scalar-valued quantity, yet the mesh structure is mostly lost.

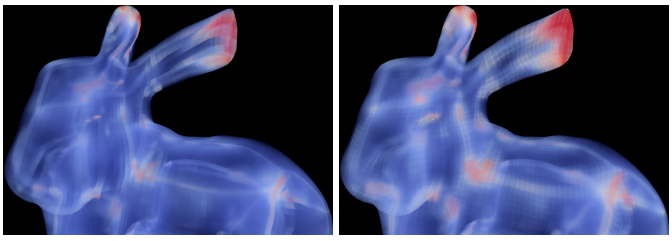

Fig. 7. Left: Volume rendering using cell contributions. Right: Face-based volume rendering. The Jacobian ratio (from low to high) is mapped linearly to color (from blue to red) and opacity. Model bunny courtesy of [1].

To also accentuate the mesh structure in the context region, we refrain from using direct volume rendering. Instead, the faces are blended in correct visibility order, yet the optical depth through the cells is neglected and face colors are blended using opacities that continually increase with decreasing distance to the focus. I.e., the face opacity $\alpha_f$ is computed by modulating a user selected

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2021.3074607, IEEE Transactions on Visualization and Computer Graphics



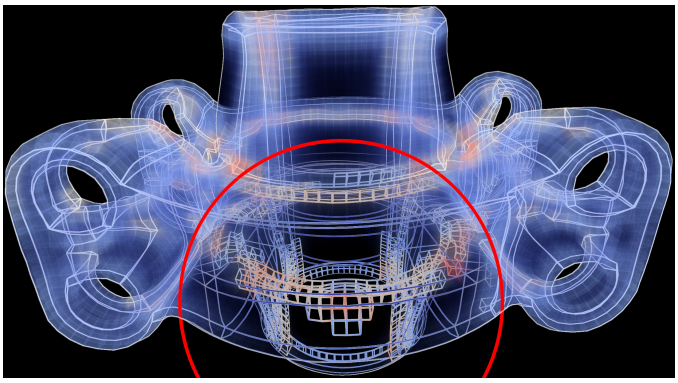

Fig. 8. Final mesh rendering shows a smooth transition from the focus edges to the context edges and volumetric representation. Model grayloc courtesy of [30].

face opacity $\hat{\alpha}_f$ using the distance to the focus center and the edge accentuation factor as

$$\alpha_f = \hat{\alpha}_f \cdot dist^4. \tag{3}$$

Blending a discrete set of faces generates accentuated jumps in the final colors whenever there is a change in the number of faces falling into adjacent fragments (Fig. 7 (right)). Increasing opacity artificially increases these jumps in the context region and makes them more noticeably. The face colors $C_f$ are generated by interpolation of per-vertex importance values by the rasterizer.

We decided to use a dark background, because the rendering, combined with bold saturated colors, tends to stand out. A white background shines through and affects the line colors. However, our visualization tool also allows switching to a white background if desired (cf. Fig. 21).

### 4.2 Blending Focus and Context

Each fragment obtains an edge and a face color $(C_e, C_f)$, and in addition computes the values $\alpha_e$, $s_e$ and $\alpha_f$ according to Eqns. 1, 2 and 3. The fragment shader blends the edge colors (focus and contextual edges) and face colors (face colors and accentuated lines) according to

$$\begin{aligned} C &= \alpha_e C_e + (1-\alpha_e) s_e \alpha_f C_f \\ \alpha &= \alpha_e + (1-\alpha_e) s_e \alpha_f. \end{aligned} \tag{4}$$

Thus, focus and context information is blended as shown in Fig. 9. Via front-to-back $\alpha$-compositing, all fragments falling into a pixel are finally merged. The context factor has a lower slope for lower distances in order to get a smoother transition in the beginning than for the sharper focus edges.

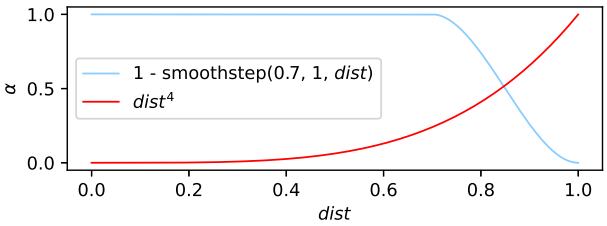


Fig. 9. Blend factors for focus (blue) and context (red).

Fig. 8 shows the final F+C look. An accentuated edge in the context takes on the color of the face, brightened a little, and its opacity is increased by about 50%. In addition, white exterior and interior screen-space silhouettes are added to improve the perception of the mesh shape [31], [32], [33]. Therefore, the mesh boundary surface is rendered, and fragments along sharp edges in the depth buffer are emphasized.

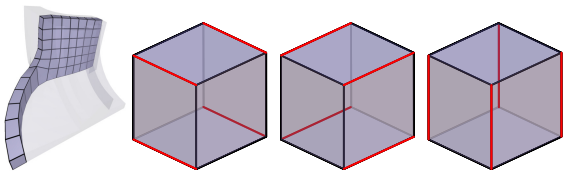

Fig. 10. A hexahedral sheet (left), and the three sets of topologically parallel edges (in red) of a hexahedral element [10]. Model courtesy of [35].

## 5 Level of Detail Structure

In the following, we describe the construction of the LoD edge structure for a given hex-mesh using topological simplification. Our approach builds upon the concept of hexahedral sheets. Hexahedral sheets were introduced by Borden et al. [11], and further formalized by [10] as a set of hex-elements which are connected to each other via their topologically parallel shared edges. In Fig. 10, we reproduce images from Woodbury et al. to illustrate the relationship between these two topology-based groups. In a number of works, the concept of hexahedral sheets has been utilized for hex-mesh construction and simplification [34], as well as re-meshing [12]. We make use in particular of sheet-based topology simplification, by successively collapsing pairs of neighboring sheets.

We use the approach proposed by [10] to extract each single sheet: Upon selecting the start edge, all elements incident to the edge are found and added to the sheet (if not done already). For each of the newly added elements, the three edges topologically parallel to the original edge are determined, and the edges are updated with the newly found edges. This process is repeated until there is no new element found. During the extraction of a single sheet, all visited element edges are recorded. Then, an unvisited edge is selected for computing a new sheet until no such edge is left. In this way, the set of sheets covering the entire hex-mesh is extracted. Finally, we define for each sheet a sheet component consisting of all elements belonging to this sheet.

### 5.1 Merging Sheet Components

In an iterative process, pairs of sheet components are merged into a joint component until no components can be merged anymore. Therefore, for all pairs of sheet components, their neighborhood relation is classified analogously to the work by [6] as

- adjacent (or tangent),
- intersecting,
- hybrid (i.e., tangent and intersecting),
- none.

Fig. 11 illustrates the different constellations. In our design, sheet components are neighbors only if they share at least one boundary face that is no longer on the boundary after merging.

In addition to the neighborhood relation, for each pair of neighboring components a weight is computed. The weights are used in an iterative merging process to determine the priority of merging for each neighboring component pair. Building upon [6], where the weights consider the percentage of merged boundary faces to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2021.3074607, IEEE Transactions on Visualization and Computer Graphics



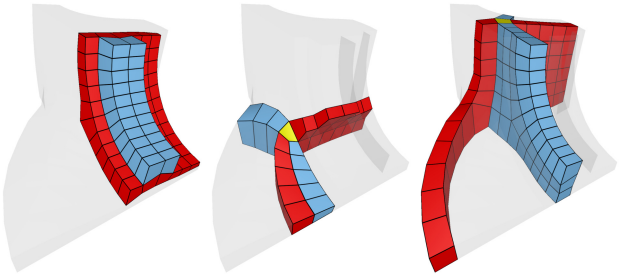

Fig. 11. From left to right, the different topological relations (adjacent, hybrid, intersecting) of neighboring hexahedral sheets. Similarity to the constellations by [6] is intentional. Model courtesy of [35].

the overall number of boundary faces in the two components, the weights are computed as

$$w_{i,j} = \frac{\partial C_i \cap \partial C_j}{|\,\partial C_i\,| + |\,\partial C_j\,|} \cdot \frac{1}{|\,C_i\,| + |\,C_j\,|} \tag{5}$$

Here, $\partial C_i \cap \partial C_j$ is the number of boundary element faces shared by the pair of neighboring components $C_i$ and $C_j$, and $|\,C_i\,| + |\,C_j\,|$ is the number of cells in $C_i$ and $C_j$. Different to [6], the weights consider the topological size (i.e., the number of cells) for merging to reduce the potential 'jumps' in the LoD structure, i.e., neighboring pairs with smaller topological sizes are favoured at similar ratio between boundary faces. Even though we favour a purely topological measure in this work, alternatively one could also opt to use the face areas and cell volumes.

The adjacency information is stored in a priority queue, with the weights serving as the priority measure. Pairs of components with highest priority are merged first, yet adjacent sheets always have a higher priority than hybrid sheets, and hybrid sheets always have a higher priority than intersecting sheets. During merging, the two matching components are removed from the component queue, and a new component is inserted. The edges on the shared boundary faces of these components are identified and marked as invisible on this level (Fig. 12). Then the side element faces of the new component are recomputed, and the adjacency information as well as the priority of neighboring components is updated in the component queue. A next coarser LoD level is established as soon as the number of cells of the merged component is at least more than twice as large as the number of cells of the (merged) components on which the last LoD level starts. The merging process is repeated until only one single component is left.

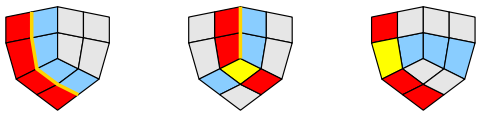

Fig. 12. Sheet neighborhoods in a 2D quad mesh. Bold orange lines become invisible after merging. From left to right: Adjacent sheets, hybrid sheets, intersecting sheets. No edges become invisible when sheets intersect.

An exception to the rules is made for so-called singular edges. Singular or irregular edges are those edges which do not have exactly 2 (on the boundary) or 4 (in the interior) incident cells [34]. These edges form curves which separate the hex-mesh into its regular parts, and they serve as important visual cues regarding the global mesh topology. In particular, valence 1 edges are never set to be invisible, and singular edges of all other valences are only invisible at the coarsest LoD level.

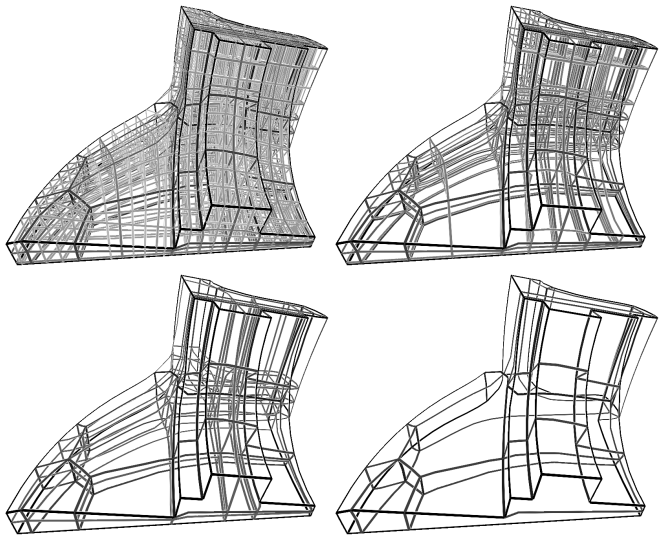

Fig. 13. From left to right: LoD levels 0, 2, 3 and 4 of a hex-mesh. Model fandisk courtesy of [27].

Fig. 13 and Fig. 14 show the extracted LoD structures of two hex-meshes. The former shows the model from Fig. 4, yet now the edges at different LoD levels, i.e., with $e_{level}$ equal to 0, 2, 3, and 4, are shown separately to better demonstrate the sequence of merging steps. The same representation is used for the latter examples, yet the edges with $e_{level}$ equal to 0, 3, 5 and 6 are shown. In both cases, the greyscale encoding of LoD levels as in Fig. 4 is used.

### 5.2 LoD Symmetry

An interesting question is whether the LoD construction process maintains certain properties of the initial mesh, such as symmetry. When consecutively merging multiple topologically symmetric pairs of sheet components, the symmetry usually remains intact in the LoD edge structure. This is due to the used LoD transition strategy, where the next coarser LoD is only started when the number of merged cells is two times larger than the number of cells the last LoD level was started with. However, multiple properties must be fulfilled.

First of all, a pair of sheet components that is merged must not have a higher merging priority with neighboring components after merging than another pair of sheet components symmetric to this pair. Otherwise, symmetric pairs of components might be missed, and the construction process advances to the next LoD level before these pairs are merged. There is no guarantee for this to be true, but the chance that this situation occurs is significantly decreased by the merging process according to Eqn. 5. As merged components in general have a higher number of cells, and thus a lower merging priority, the approach didn't result in any failure in the examples we have used.

Secondly, if we have symmetry with an uneven number of topologically symmetric sheet components, the symmetry property can be lost after merging. This is because the algorithm always merges pairs and not, for example, triples of sheet components. If there is an even number of more than two topologically symmetric sheet components next to each other in a periodic topology, the order in which they are merged is arbitrary and dependent on the order of the edges in the mesh data. In order to make sure

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2021.3074607, IEEE Transactions on Visualization and Computer Graphics



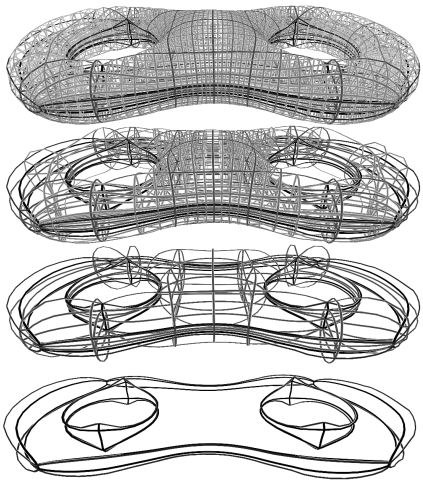

Fig. 14. From top to bottom: LoD levels 0, 3, 5 and 6 of a hex-mesh. Model eight courtesy of [36].

that symmetry is not lost, a perfect matching strategy for sheet component pairs with equal matching weights is used instead of consecutive greedy matching. For example, the symmetric sheet components (A, B, C, D, E, F) are then guaranteed to be merged to, e.g., (AB, CD, EF), while otherwise they might get merged to (AB, C, DE, F) and symmetry is broken.

## 6 GPU Implementation

Our reference implementation uses the functionality provided by OpenGL 4.5. All data required for rendering is kept on the GPU, so that no CPU-GPU communication is required during rendering and user interaction. Since the fragment shader always performs all computations described in Sec. 4, the user can arbitrarily change the size of the focus lens without affecting performance. All constant parameters in Eqns. 1, 2 and 3 are issued via constant shader parameters that can be changed interactively by the user.

In order to make the single-pass face-based rendering of faces and edges possible, we use *programmable vertex pulling* [37]. We use a variant called *programmable attribute fetching*, where a fixed-function element array is used for indexed primitive rendering, but all vertex attributes are loaded manually from a dedicated buffer. For each cell face, we create two triangles with shared vertices only between these two triangles. Then, by using the vertex ID the fragment shader computes which face a vertex belongs to, and loads the correct face data. A geometry representation where all vertices are shared between faces is not possible, as vertices need to pass different data to the fragment shader depending on the current face. Thus, the renderer cannot utilize the post-transform cache of indexed vertices between faces, letting the pure geometry throughput fall slightly below the GPU limit.

The fragment shader uses the vertex positions of all four face corner points to compute the shortest distance to any of the face edges. When rendering edges with per-edge constant color, star-shaped patterns occur at edge intersections (Fig. 15a). Since smoothly interpolated per-vertex colors are rendered, these patterns are hardly visible (Fig. 15b). Only when two edges intersect and one is not rendered (Fig. 15c), the pattern is clearly visible. This is avoided by letting the shader ignore edges in the distance calculation which are not visible (Fig. 15d).

To keep track of the fragments falling into the same pixel, we employ GPU per-pixel linked lists [18]. All generated fragments

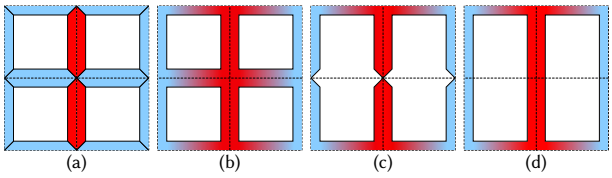


Fig. 15. Edge rendering in 2D. (a) Four edges meet in one vertex and form an arrow-like shape. (b) We linearly interpolate colors in order to make the arrow-like shapes disappear. (c) Making edges invisible creates holes. (d) An approach for closing these holes is ignoring lines with a low opacity in the calculation of the closest edge.

are stored in a linked list over all pixels, and a fragment shader sorts these fragments w.r.t. their screen space depth. Here it is assumed that the GPU buffers used for storing the fragments along with a reference to the next neighbor in the global fragment list are large enough. We demonstrate in Sec. 7 that even for hex-meshes with a few million elements this is case. For scenes with high depth complexity, however, the number of fragments is so large that sorting can become a performance bottleneck. For instance, for the largest hex-mesh used in our experiments about 340 million fragments are generated per frame. Therefore, we use a GPU version of priority-queues using a binary tree implementation as search structure [25], which reduces the time required for sorting to slightly more than half of the overall frame time.

## 7 Results and Analysis

All our results were rendered on an NVIDIA RTX 2070 GPU with 8GB of on-chip memory. Only the construction of the LoD hiearchy was performed on the CPU, i.e., a workstation running Ubuntu 20.04 with an AMD Ryzen 9 3900X @3.80GHz CPU and 32GB RAM. We use different viewport sizes to demonstrate the scalability of the rendering approach in the number of pixels, and in particular to show that even for large hex-meshes and viewports the memory required by per-pixel fragment lists does not exceed the GPU memory. All timings are averages over 128 frames with different camera views where the data sets cover almost all of the screen. The accompanying video shows one of the camera paths we have used to record the performance data.

Table 1 lists the number of hex-elements of each of the test data sets, the GPU memory that is required to store these data sets on the GPU, and the time required to build the LoD structures. We have in particular included the data sets ”example3” and ”cubic128” (Fig. 20) to demonstrate that even large data sets with millions of cells can be stored entirely on the GPU and processed in a short time.

| Data Set | #Cells | Mesh Buffer Size | LoD Creation Time |
|---|---|---|---|
| fandisk | 1,774 | 0.5 MiB | 0.01s |
| eight | 5,428 | 1.4 MiB | 0.05s |
| dragon | 14,009 | 3.5 MiB | 0.2s |
| grayloc | 24,344 | 5.9 MiB | 0.4s |
| armadillo | 29,935 | 7.2 MiB | 0.5s |
| anc101_a1 | 73,976 | 17.3 MiB | 2.1s |
| example3 | 589,040 | 135.9 MiB | 16.3s |
| cubic128 | 2,097,152 | 476.5 MiB | 23.0s |

TABLE 1
Data set statistics. Model fandisk courtesy of [27], eight courtesy of [36], dragon, armadillo and dancingchildren courtesy of [29], grayloc courtesy of [30], anc101_a1 courtesy of [1], cognit courtesy of [3], example3 courtesy of [38], model cubic_128 is a twisted Cartesian grid of size $128^3$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2021.3074607, IEEE Transactions on Visualization and Computer Graphics



| Data Set | Viewport | FPS | F+C | PPFL | Mem. PPFL |
|---|---|---|---|---|---|
| grayloc | 1280x720 | 154 FPS | 2.2ms | 4.3ms | 0.21 GiB |
| | 1920x1080 | 85 FPS | 3.9ms | 7.8ms | 0.47 GiB |
| | 2560x1440 | 52 FPS | 6.5ms | 12.7ms | 0.84 GiB |
| anc101_a1 | 1280x720 | 96 FPS | 4.1ms | 6.3ms | 0.37 GiB |
| | 1920x1080 | 51 FPS | 6.7ms | 12.9ms | 0.83 GiB |
| | 2560x1440 | 32 FPS | 11.0ms | 20.3ms | 1.48 GiB |
| cognit | 1280x720 | 127 FPS | 3.2ms | 4.6ms | 0.20 GiB |
| | 1920x1080 | 74 FPS | 4.6ms | 8.8ms | 0.45 GiB |
| | 2560x1440 | 49 FPS | 6.6ms | 13.8ms | 0.80 GiB |
| example3 | 1280x720 | 44 FPS | 13.0ms | 9.8ms | 0.64 GiB |
| | 1920x1080 | 26 FPS | 17.8ms | 20.6ms | 1.45 GiB |
| | 2560x1440 | 17 FPS | 23.6ms | 33.6ms | 2.57 GiB |
| cubic128 | 1280x720 | 11.9 FPS | 37.3ms | 46.9ms | 1.19 GiB |
| | 1920x1080 | 6.8 FPS | 45.8ms | 101.5ms | 2.68 GiB |
| | 2560x1440 | 1.8 FPS | 152.7ms | 196.8ms | 1.19 GiB* |

TABLE 2
Performance statistics for selected data sets at different viewport sizes: Frames per second (FPS), times required by the F+C rendering stage (F+C) and the stage that sorts and blends the fragments in the per-pixel fragment lists (PPFL), and the memory consumed by the fragment lists (Mem. PPFL). In the last row, the viewport was split into 4 tiles of 1280x720 and the data set was rendered consecutively into each tile. Rendering to the full viewport requires 4.8 GiB and is not possible due to OpenGL buffer restrictions to 4 GiB.

| Data Set | Viewport | FPS | Mem. PPFL | Mem. Geom. |
|---|---|---|---|---|
| parts1 | 1280x720 | 9.5 FPS | 0.44 GiB | 2.09 GiB |
| | 1920x1080 | 5.4 FPS | 0.98 GiB | |
| | 2560x1440 | 3.4 FPS | 1.75 GiB | |
| bumpy_torus | 1280x720 | 4.6 FPS | 0.68 GiB | 4.27 GiB |
| | 1920x1080 | 2.5 FPS | 1.52 GiB | |
| | 2560x1440 | 1.7 FPS | 2.71 GiB | |

TABLE 3
Performance statistics for rendering numerical simulation grids with 24 (parts1) and 50 (bumpy_torus) million hexahedral finite elements using face-based volume rendering: Frames per second (FPS), the memory consumed by the fragment lists (Mem. PPFL), and the memory consumed by the geometry data (Mem. Geom.).

Table 2 provides a performance statistics, distinguishing between the rendering stage used to determine the F+C look, and the rendering stage that sorts and blends the fragments in the fragment lists. In addition, the memory requirements of the fragment lists are given. It can be seen that the advantage of not having to store any per-cell adjacency information to render the elements in correct visibility order comes with additional computational load and memory requirements for handling the fragment lists. For the largest data set with 2 million elements, the frame rate drops down to 6.8 fps at Full HD screen resolution. It can further be seen that the fragment shader consumes the vast amount of the total frame time. The time for resolving the per-pixel fragment lists is between 43% and 72% of the total rendering time, and it is dependent on the depth complexity of the data set, i.e., the number of cells falling into the single pixels.

At high screen resolution, the fragment lists can become so large that the OpenGL buffer restriction of 4 GiB is reached. This can be overcome by subdividing the screen into disjoint regions and rendering to each region sequentially in multiple rendering passes (see last row of Table 2). This approach requires processing each cell multiple times in the geometry and rasterization stage, but it does not increase the number of fragment shader operations and effectively reduces the required GPU memory. By splitting the 2560x1440 viewport into four tiles of size 1280x720, cubic128 can then be rendered with roughly 1.8 fps.

Screen-tiling also gives rise to an efficient implementation of a sort-first parallelization strategy when using a multi-GPU cluster. In sort-first, each GPU holds the entire mesh and renders into a disjoint screen region, so that only the corresponding fraction of the per-pixel linked list is required on each GPU. If also the geometry buffers storing mesh vertices and indices exceed the buffer limit, theses buffers can be tiled as well and rendered consecutively. Anticipating that merging screen tiles to form the final image requires by far less time than rendering, this approach scales effectively in the number of cells. Furthermore, it enables to visualize very large data sets on current clusters comprising multiple high-end GPUs with up to 24GB on-chip memory.

In the following, we show results of interactive visual inspections of some of the test data sets using the proposed F+C renderer. In all examples, the per-cell scaled Jacobian ratio is mapped to color (from blue to red) and opacity (from 0 to 1). Fig. 18 and Fig. 19 show the use of F+C rendering to obtain an overview of the spatial locations of regions with highly deformed cells, and to select a particular focus region for a more detailed analysis. One can see that due to the combination of contextual lines with volumetric face-based rendering and accentuated edges, the user quickly understands the basic structure of the mesh and its subdivision into multiple regular components. Once a focus region is selected, a detailed analysis of the cells in that region is performed via close-up views and interactive navigation. During inspection, LoD levels, transfer functions for edge and face colors and opacities, as well as edge thickness can be changed interactively to enhance the visual representation.

Fig. 20 (left) shows a deformed Cartesian grid comprised of $128^3$ cells. The deformed grid is created by performing a Finite Element analysis with a specific boundary condition to let the mesh twist. High deformations occur in the orange regions, yet the cells are so small that the mesh structure cannot be seen. Via the edges from a selected coarse LoD level, the basic mesh structure is preserved, and the user can now zoom at a high deformation region and use focus rendering to investigate the deformations in more detail. Fig. 20 (right) shows a rendering of a hex-mesh that was generated via the method from [38]. As can be seen, the meshing approach creates many singular edge columns, i.e., cells with higher deformations are laid out along straight vertical structures, while the remaining parts of the mesh show almost zero deformation. The focus view reveals the structure of the cells with a deformation larger than a selected threshold in the selected region.

To further demonstrate the capabilities of our tool regarding task T4, we use two large simulation data sets comprising 24 and 50 million elements, respectively (see Table 3. Both data sets have been generated by discretizing the interior of a surface model into hexahedral finite elements, which are then used in an elasticity simulation w.r.t. external loads [39]. Due to the applied loads, the simulation elements deform. The occurring stresses (using the scalar von Mises stress norm) are assigned to each vertex as an additional attribute. Rendering the meshes is performed using face-based volume rendering (see Fig. 26, 25% faster than F+C rendering), using a 2D color table to distinguish between regions with only deformation, only high stress, or both. For the 50M elements data set, the geometry data consumes more than 4 GiB of memory, but since the data is split into separate vertex and index buffers the 4 GiB OpenGL buffer limit is not reached. At the largest viewport resolution of 2560x1440, the two data sets render at frame rates of 3.4 and 1.7 FPS, respectively. Note here that due to the aspect ratio of the data set, about 1/4 of the pixels are not covered and, thus, the fragment lists remain under the 4 GiB limit.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2021.3074607, IEEE Transactions on Visualization and Computer Graphics



### 7.1 Evaluation

We performed an informal expert evaluation with the goal to obtain feedback concerning the strengths and weaknesses of our tool compared to the one by Bracci and co-workers [5]. We let two geometric modelling experts, two experts from computational science, and two experienced computer graphics students work with both tools. With each tool, the users visualized three different hex-meshes. The users were asked to comment on how effectively they understand the overall shape of the objects, determine the regions with highly deformed cells, and assess the spatial relationships between regions with different deformation strengths and the concrete deformation characteristics of cells in regions containing highly deformed cells. Visual comparisons to HexaLab [5] and the main sheet extraction method of Xu et al. [6] are given in Fig. 22, 23, 24 and 25. In the expert evaluation we did not consider the method by Xu et al., since its focus is on a topological mesh analysis and not on the visualization of cell deformations. The major findings are as follows:

**Global view** Experts appreciate that the global context is always visible when using our tool. Due to the use of face-based volume rendering with deformation strength-based classification in the context region, all regions with high deformation cells can be perceived in relation to each other (T1). The visualization hints at all potentially interesting regions. With only face-based volume rendering, however, users sometimes loose the depth perception and feel that the global mesh structure cannot be understood well. This limitation is remedied by blending coarse-scale edge structures into the context region, which enhances the understanding of the global mesh structure without introducing clutter (T3). HexaLab, in comparison, supports the rendering of singular edges in filtered mesh regions and a transparent mesh outline to maintain some global context. Users perceived as a minor limitation the resulting sparseness of regions in which cells are filtered out (cf. Fig. 22).

**LoD structure** Unlike when using slicing, peeling and quality-based cell filtering, where cells are removed entirely based on a binary decision criterion, users appreciate the smooth LoD-based transition from focus to context and high to low deformations provided by our tool. This effectively reveals how the cell quality changes globally, and whether these changes are rather smooth or occur abruptly (T1).

**Edge-based rendering** When inspecting regions via the screen space lens and edge rendering, users were able to quickly access both the embedding of deformed cells into the surrounding and the variations of the cell deformations (T2). Both was deemed difficult with opaque face rendering, since only the front-most faces are shown and context information is increasingly removed.

**Scalar field visualization** The two users from computational science found it appealing that also scalar values given per vertex or cell can be visualized in turn using face-based volume rendering (T4). In particular when hex-meshes are used as simulation grids, the option to visualize the relationships between simulation accuracy and deformation of simulation cells provides new insights into the simulation.

**Interactive visual parameter modification** It was perceived very supportive of a detailed mesh analysis that all rendering parameters could be changed interactively, and, thus, groups of elements could be quickly (de-)emphasized while enabling less and more attention on the global mesh structure.

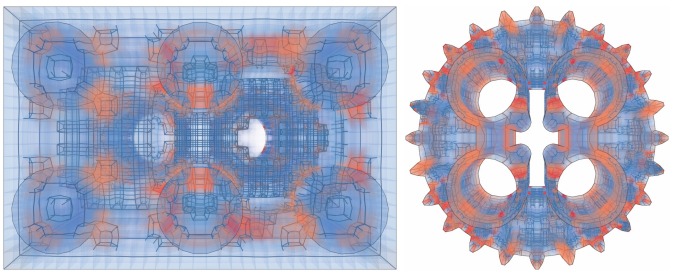

Fig. 16. Contextual volume rendering of meshes from an octree-based meshing approach. Due to the highly irregular topological structure, many short line segments are created. Models courtesy of [40].

### 7.2 Limitations and improvements

Among the experts there was consensus regarding some general limitations of the proposed F+C technique:

**Deformation assessment** In cases where the variation of the geometric cell deformations is high, some users found it difficult to grasp the deformations effectively using edge-based rendering. In particular, the assignment of edges to cells could not be accessed accurately (see Fig. 22). In these cases, users wished for the possibility to alternate between edge and cell rendering to further support a fine-granular inspection of the cell clusters. We have included this rendering option into the tool, in combination with a smooth fading of opaque faces with increasing distance to the focus center (see Fig. 8, 19, 20, 22).

**Occlusions** Since all elements along the viewing cone are put into focus when using a screen space lens, in some situations clutter was perceived and some important regions were occluded. To overcome this limitation, we have added an object space lens (cf. Fig. 19). The user picks a pixel, and the lens is automatically centered at the closest mesh point along the viewing ray through that pixel. The lens can then be moved along this ray into the object using the mouse wheel. When the user moves the camera, the object space center-point of the lens remains unchanged.

**LoD construction** When a meshing technique produces meshes with a very high number of singularities (e.g., octree-based meshing techniques, cf. Fig. 16), the LoD structure is cluttered as well and becomes less useful. This drawback also affects tools like HexaLab [5], which renders singular edges in regions where cells are filtered out. Xu et al. [6] also state regarding their method that “it is still hard to show the structure of an octree or tet-split hex-mesh due to the overly complex structure and a large number of extracted main sheets”. Similarly to octree-based meshing techniques, meshes cut out of Cartesian grids also produce a large amount of singular edges on the mesh boundary. Anisotropic meshes, on the other hand, like meshes with adapted scaling close to boundaries for CFD simulations, can be handled well (cf. Fig. 17).

## 8 Conclusion and Future Work

In this paper, we have introduced an interactive F+C rendering technique for hex-meshes using fragment-based edge and face rendering. We have demonstrated that even high-resolution meshes can be rendered at high visual quality by using a carefully designed combination of detailed cell information and surrounding contextual information. To achieve this, we have introduced the use of hexahedral sheets for extracting a hierarchical LoD edge structure that provides important shape cues in the context region. This allows reducing occlusions and visual clutter. By using

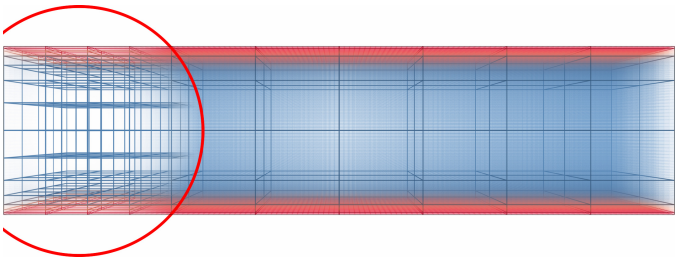

Fig. 17. F+C rendering of a grid mesh (128x128x32) with tanh stretching in y direction near simulation boundaries. The LoD construction algorithm can also handle anisotropic meshes well.

a purely fragment-based rendering approach, which smoothly transitions between highly detailed edge rendering and volumetric face blending, interactive rendering of data sets comprised of up to a few millions of elements is achieved on current GPU architectures. Our results indicate the potential of the proposed rendering technique for an interactive visual inspection of hex-meshes, supported by an automated guidance to important mesh regions.

In the future, we will consider the integration of approximate rendering techniques for transparent fragments that can avoid the use of per-pixel fragment lists (see [25] for an overview) Such techniques do not render the fragments in correct visibility order, yet since our approach uses transparency mostly in the context region with more emphasis on closer mesh structures, they might be able to provide a meaningful approximation. Additionally, we will investigate the implementation of a distributed memory sort-first parallelization strategy on a multi-GPU cluster to visualize very large meshes. We further envision an AR-based stereoscopic inspection of hex-meshes to provide an improved spatial understanding of shape variations. Here it will be interesting to analyse whether a purely fragment-based approach is suitable for stereoscopic rendering. Furthermore, we will investigate means to visualize topologically irregular hex-meshes, e.g., as generated by octree-based techniques, to avoid the resulting clutter. Finally, we intend to extend the rendering technique to perform volume rendering of physical fields given at the hexahedral cells or vertices. This includes in particular the use of extended barycentric interpolation for deformed hex-cells and the rendering of implicit isosurfaces going through the cells.

## 9 Acknowledgments

The authors would like to thank Maximilian Bandle, Technical University of Munich, for his support concerning the efficient implementation of per-pixel fragment sorting on GPUs, and the various authors for providing the hex-meshes we have used. This work was partially funded by the German Research Foundation (DFG) under grant number WE 2754/10-1 “Stress Visualization via Force-Induced Material Growth”.

## References

[1] J. Gregson, A. Sheffer, and E. Zhang, “All-hex mesh generation via volumetric polycube deformation,” *Computer Graphics Forum (Special Issue of Symposium on Geometry Processing 2011)*, vol. 30, no. 5, p. to appear, 2011.

[2] X. Gao, D. Panozzo, W. Wang, Z. Deng, and G. Chen, “Robust structure simplification for hex re-meshing,” *ACM Trans. Graph.*, vol. 36, no. 6, Nov. 2017. [Online]. Available: https://doi.org/10.1145/3130800.3130848

[3] J. Huang, T. Jiang, Z. Shi, Y. Tong, H. Bao, and M. Desbrun, “$\ell$1-based construction of polycube maps from complex shapes,” *ACM Transactions on Graphics (TOG)*, vol. 33, no. 3, pp. 1–11, 2014.

[4] M. Livesu, N. Pietroni, E. Puppo, A. Sheffer, and P. Cignoni, “Loopy cuts: Surface-field aware block decomposition for hex-meshing,” *arXiv preprint arXiv:1903.10754*, 2019.

[5] M. Bracci, M. Tarini, N. Pietroni, M. Livesu, and P. Cignoni, “Hexalab.net: An online viewer for hexahedral meshes,” *Computer-Aided Design*, vol. 110, pp. 24 – 36, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010448518304238

[6] K. Xu and G. Chen, “Hexahedral mesh structure visualization and evaluation,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 1173–1182, 2019.

[7] M. Gleicher, D. Albers, R. Walker, I. Jusufi, C. D. Hansen, and J. C. Roberts, “Visual comparison for information visualization,” *Information Visualization*, vol. 10, no. 4, p. 289–309, Oct. 2011. [Online]. Available: https://doi.org/10.1177/1473871611416549

[8] C. Stimpson, C. Ernst, P. Knupp, P. Pébay, and D. Thompson, “The verdict library reference manual,” 04 2007.

[9] X. Gao, J. Huang, K. Xu, Z. Pan, Z. Deng, and G. Chen, “Evaluating hex-mesh quality metrics via correlation analysis,” *Computer Graphics Forum*, vol. 36, no. 5, pp. 105–116, 2017.

[10] A. C. Woodbury, J. F. Shepherd, M. L. Staten, and S. E. Benzley, “Localized coarsening of conforming all-hexahedral meshes,” *Eng. Comput. (Lond.)*, vol. 27, no. 1, pp. 95–104, 2011. [Online]. Available: https://doi.org/10.1007/s00366-010-0183-9

[11] M. J. Borden, S. E. Benzley, and J. F. Shepherd, “Hexahedral sheet extraction,” in *IMR*, 2002.

[12] R. Wang, C. Shen, J. Chen, H. Wu, and S. Gao, “Sheet operation based block decomposition of solid models for hex meshing,” *Computer-Aided Design*, vol. 85, pp. 123–137, 2017.

[13] C. T. Silva, J. L. D. Comba, S. P. Callahan, and F. F. Bernardon, “A survey of gpu-based volume rendering of unstructured grids,” *Revista de informática teórica e aplicada. Porto Alegre, RS. Vol. 12, n. 2 (out. 2005), p. 9-29*, 2005.

[14] M. Weiler, M. Kraus, M. Merz, and T. Ertl, “Hardware-based ray casting for tetrahedral meshes,” in *IEEE Visualization, 2003. VIS 2003.* IEEE, 2003, pp. 333–340.

[15] S. P. Callahan, M. Ikits, J. L. D. Comba, and C. T. Silva, “Hardware-assisted visibility sorting for unstructured volume rendering,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 11, no. 3, pp. 285–295, 2005.

[16] R. Marroquim, A. Maximo, R. Farias, and C. Esperança, “Gpu-based cell projection for interactive volume rendering,” in *2006 19th Brazilian Symposium on Computer Graphics and Image Processing*. IEEE, 2006, pp. 147–154.

[17] J. Georgii and R. Westermann, “A generic and scalable pipeline for gpu tetrahedral grid rendering,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1345–1352, 2006.

[18] J. C. Yang, J. Hensley, H. Grün, and N. Thibieroz, “Real-time concurrent linked list construction on the gpu,” in *Proceedings of the 21st Eurographics Conference on Rendering*, ser. EGSR’10. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2010, pp. 1297–1304.

[19] L. Carpenter, “The a-buffer, an antialiased hidden surface method,” in *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, 1984, pp. 103–108.

[20] M. Hadwiger, A. K. Al-Awami, J. Beyer, M. Agus, and H. Pfister, “Sparseleap: Efficient empty space skipping for large-scale volume rendering,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 974–983, Jan 2018.

[21] S. Marchesin, C.-K. Chen, C. Ho, and K.-L. Ma, “View-dependent streamlines for 3d vector fields,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1578–1586, 2010.

[22] T.-Y. Lee, O. Mishchenko, H.-W. Shen, and R. Crawfis, “View point evaluation and streamline filtering for flow visualization,” in *2011 IEEE Pacific Visualization Symposium*. IEEE, 2011, pp. 83–90.

[23] J. Ma, C. Wang, and C.-K. Shene, “Coherent view-dependent streamline selection for importance-driven flow visualization,” in *Visualization and Data Analysis 2013*, vol. 8654. International Society for Optics and Photonics, 2013, p. 865407.

[24] T. Günther, C. Rössl, and H. Theisel, “Opacity optimization for 3d line fields,” *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, pp. 1–8, 2013.

[25] M. Kern, C. Neuhauser, T. Maack, M. Han, W. Usher, and R. Westermann, “A comparison of rendering techniques for 3d line sets with transparency,” *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2020.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2021.3074607, IEEE Transactions on Visualization and Computer Graphics



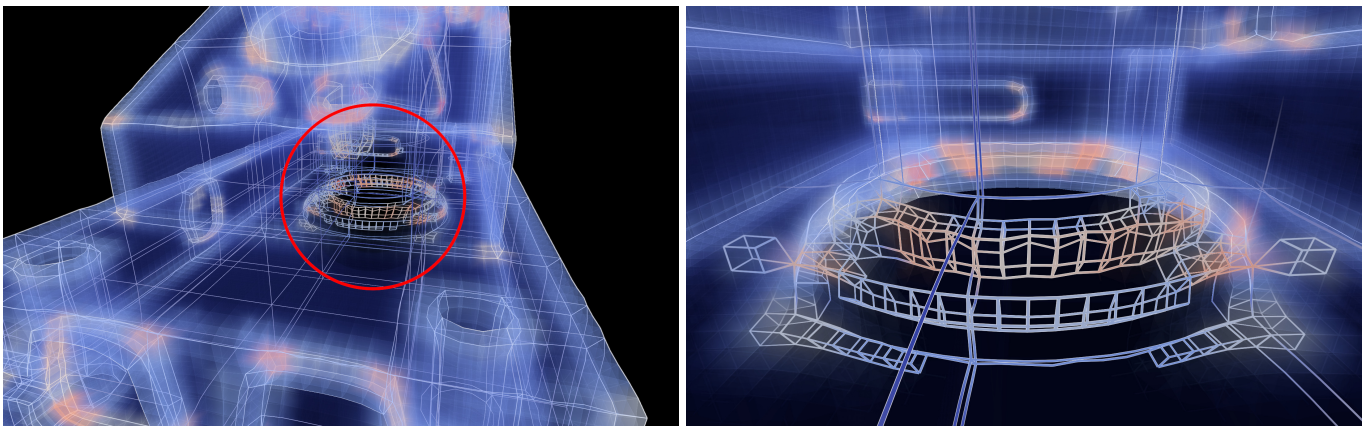

Fig. 18. The first F+C view shows a selected mesh sub-structure with highly deformed cells (framed region) in its global surrounding. Zoom-in and focus size adjustment enables a fine granular cell analysis. Surrounding cells with high deformation are still present in the context. Model anc101_a1 courtesy of [1].

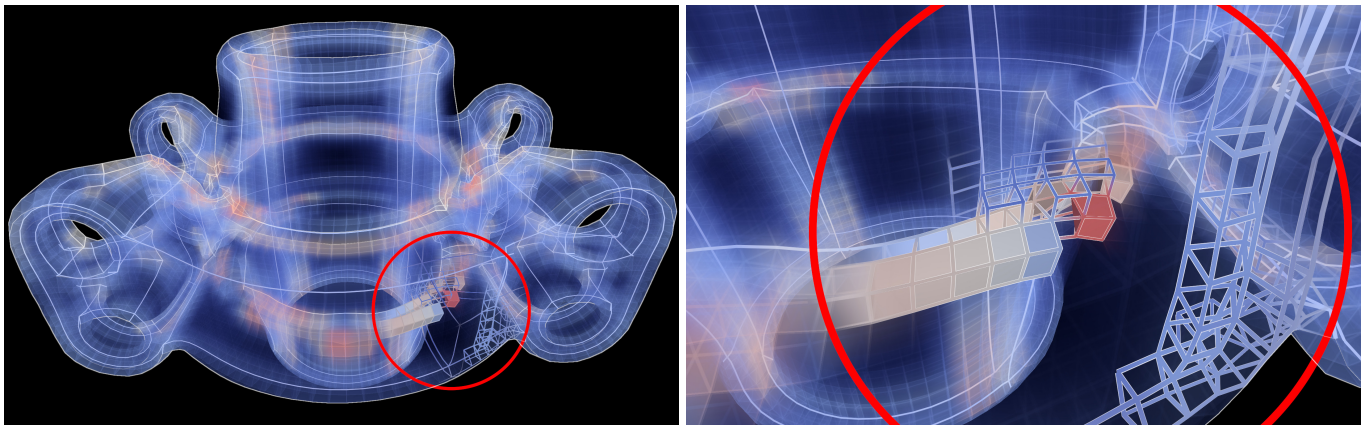

Fig. 19. In the first F+C view, important regions are effectively revealed. Framed region shows sub-structures that have been selected via the focus lens. Zoom-in and focus size adjustment enables a fine granular cell analysis. Surrounding cells with high deformation are still present in the context. Model grayloc courtesy of [30].

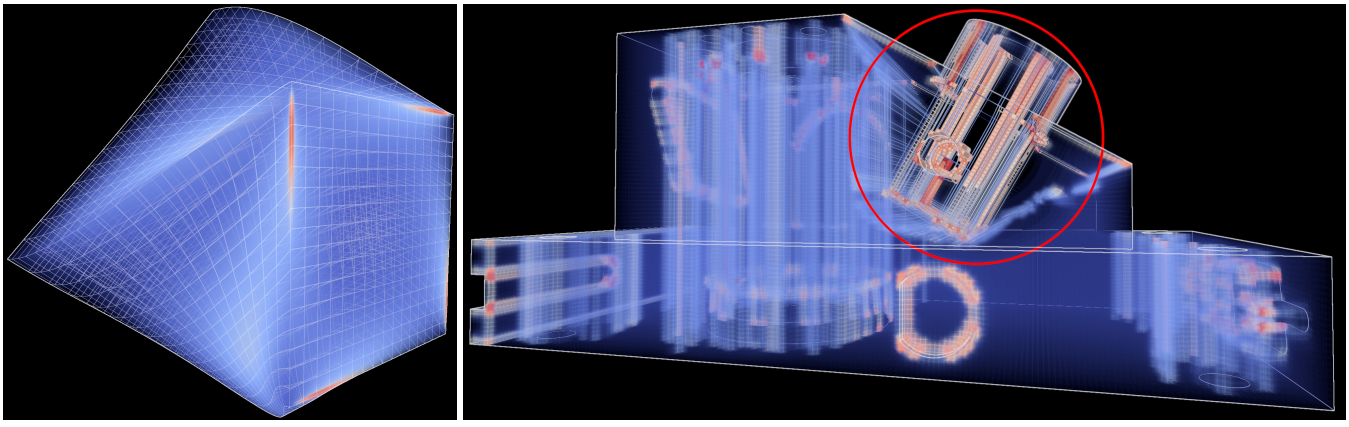

Fig. 20. Left: F+C rendering of cubic128. Right: F+C rendering of example3 reveals mostly elongated sub-structures with high deformations. Model courtesy of [38].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2021.3074607, IEEE Transactions on Visualization and Computer Graphics



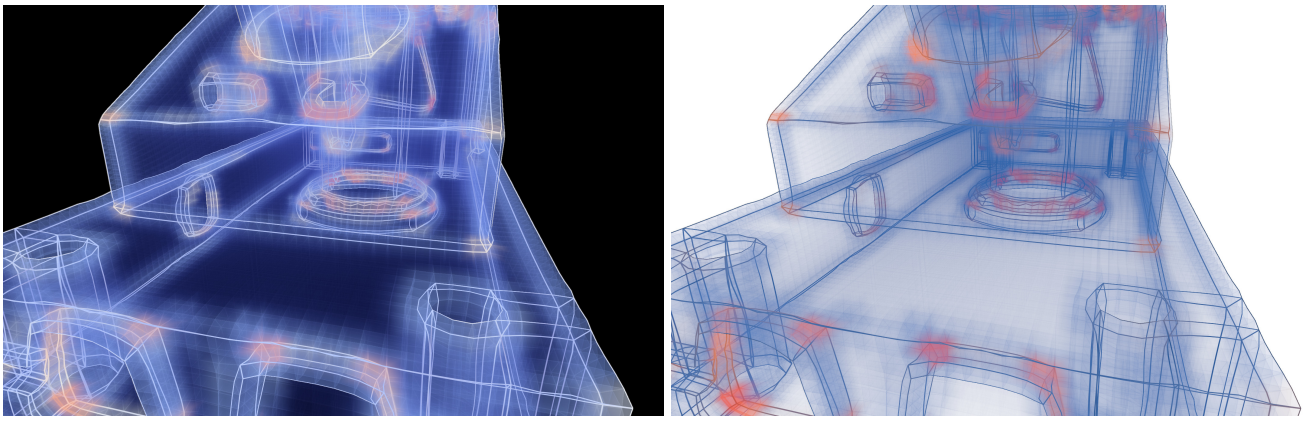

Fig. 21. Comparison of black and white background. Model anc101_a1 courtesy of [1].

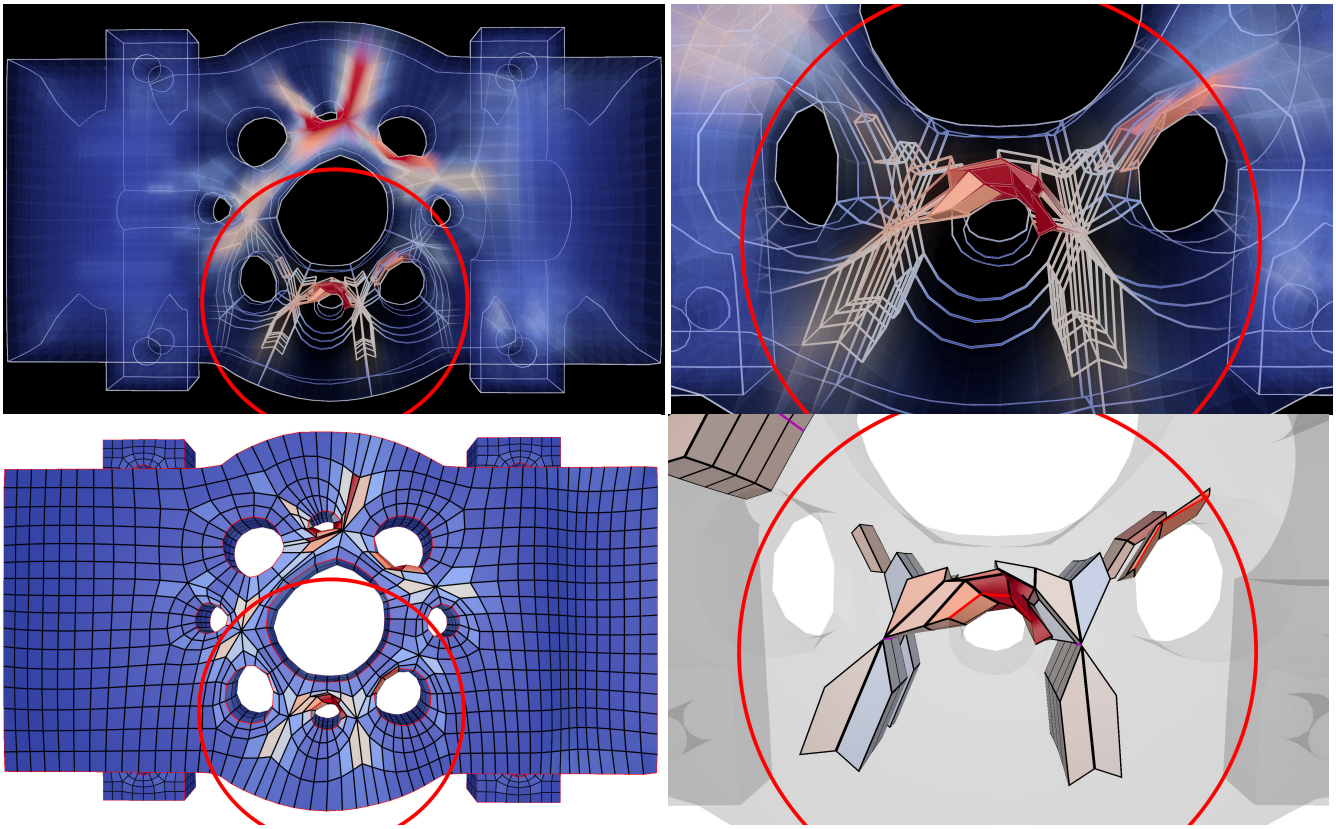

Fig. 22. Top: F+C visualization. Bottom: Same model and views with opaque surface rendering and quality-based cell filtering. Model motor_tail courtesy of [41]. Unlike the opaque surface renderer, the F+C renderer is able to retain the context around and the topological embedding of regions of interest.

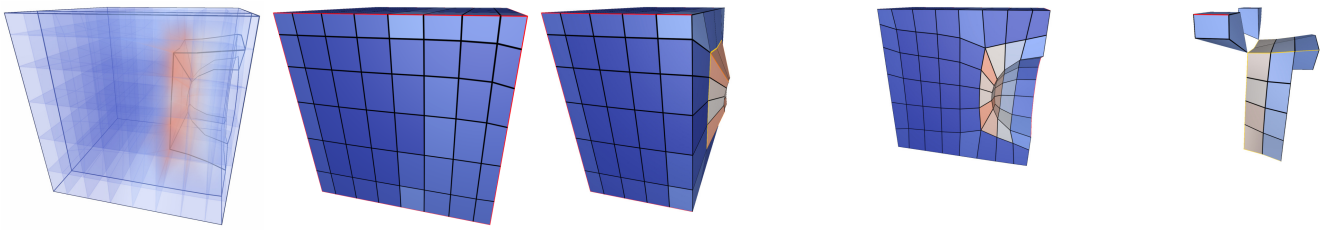

Fig. 23. From left to right: Contextual volume rendering (ours). Fully opaque surface rendering. Slicing with oblique slicing plane. Additional slicing plane removing front elements. Quality-based cell filtering. Opaque surface rendering requires multiple operations to reveal the interesting mesh structures, and context information is often lost. Model cube_carved courtesy of [41].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2021.3074607, IEEE Transactions on Visualization and Computer Graphics



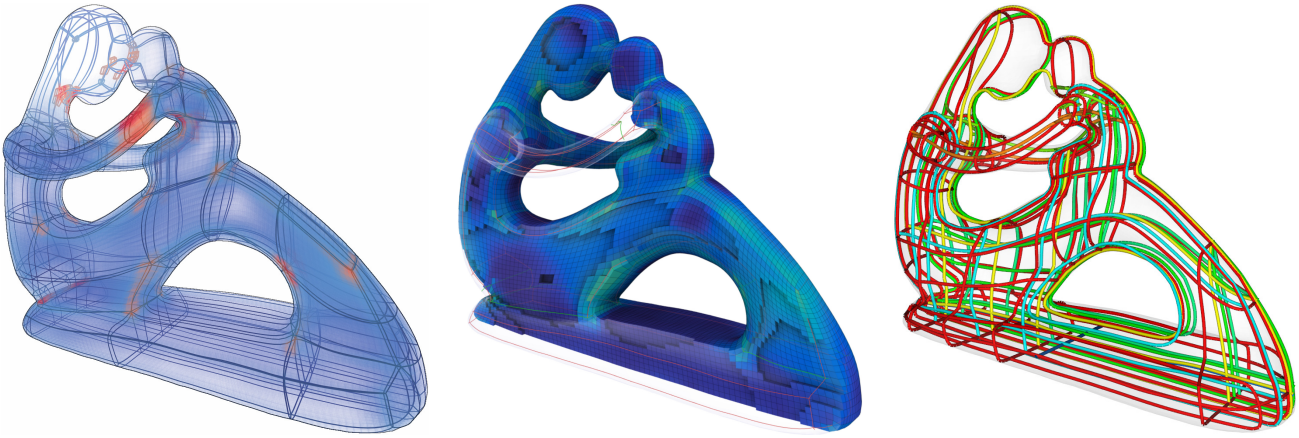

Fig. 24. Comparison of different rendering techniques. From left to right: F+C visualization with focus on woman's head (ours). Visualization in HexaLab [5] with slicing. Main sheet visualization by [6]. Rightmost image courtesy of [6]. Model fertility courtesy of [42].

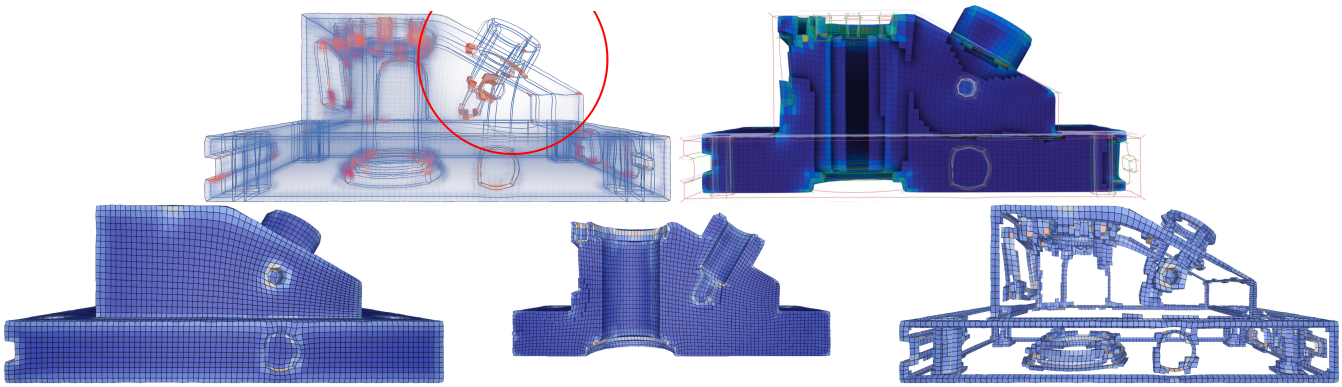

Fig. 25. From top left to bottom right: F+C visualization (ours). Slicing in HexaLab [5] with singular edges. Fully opaque surface rendering. Slicing. Quality-based cell filtering. Model anc101_a1 courtesy of [1].

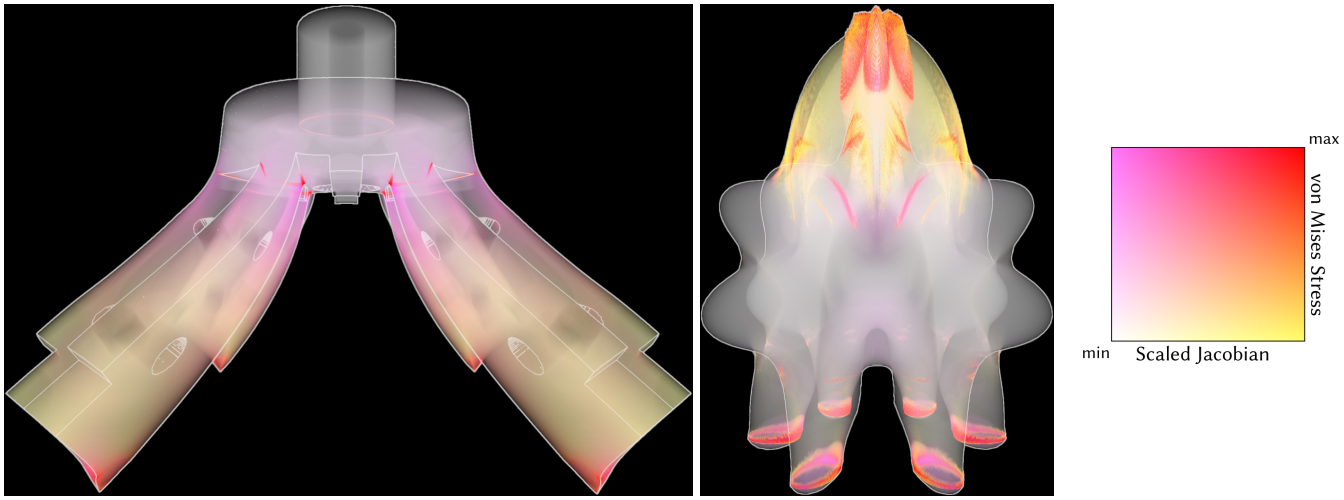


Fig. 26. Face-based volume rendering of parts1 (left, 24M cells) and bumpy_torus (middle, 50M cells) using the color map on the right.

[26] M. Kanzler, F. Ferstl, and R. Westermann, "Line density control in screen-space via balanced line hierarchies," *Computers & Graphics*, vol. 61, pp. 29–39, 2016.

[27] K. Takayama, "Dual sheet meshing: An interactive approach to robust hexahedralization," *Computer Graphics Forum*, vol. 38, no. 2, pp. 37–48, 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13617

[28] M. H. Everts, H. Bekker, J. B. T. M. Roerdink, and T. Isenberg, "Depth-dependent halos: Illustrative rendering of dense line data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1299–1306, 2009.

[29] M. Livesu, A. Sheffer, N. Vining, and M. Tarini, "Practical hex-mesh optimization via edge-cone rectification," *ACM Trans. Graph.*, vol. 34, no. 4, Jul. 2015. [Online]. Available: https://doi.org/10.1145/2766905

[30] X. Fang, W. Xu, H. Bao, and J. Huang, "All-hex meshing using closed-form induced polycube," *ACM Trans. Graph.*, vol. 35, no. 4, Jul. 2016. [Online]. Available: https://doi.org/10.1145/2897824.2925957

[31] T. Saito and T. Takahashi, "Comprehensible rendering of 3-d shapes," in *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '90. New York, NY, USA: Association for Computing Machinery, 1990, p. 197–206. [Online]. Available: https://doi.org/10.1145/97879.97901

[32] A. Hertzmann, "Introduction to 3d non-photorealistic rendering: Silhouettes and outlines," *SIGGRAPH Course Notes*, 01 1999.

[33] R. Raskar and M. Cohen, "Image precision silhouette edges," in *Proceedings of the 1999 symposium on Interactive 3D graphics*, 1999, pp. 135–140.

[34] X. Gao, Z. Deng, and G. Chen, "Hexahedral mesh re-parameterization from aligned base-complex," *ACM Trans. Graph.*, vol. 34, no. 4, Jul. 2015. [Online]. Available: https://doi.org/10.1145/2766941

[35] Y. Li, Y. Liu, W. Xu, W. Wang, and B. Guo, "All-hex meshing using singularity-restricted field," *ACM Trans. Graph.*, vol. 31, no. 6, Nov. 2012. [Online]. Available: https://doi.org/10.1145/2366145.2366196

[36] G. Xu, R. Ling, Y. J. Zhang, Z. Xiao, Z. Ji, and T. Rabczuk, "Singularity structure simplification of hexahedral mesh via weighted ranking," *CoRR*, vol. abs/1901.00238, 2019. [Online]. Available: http://arxiv.org/abs/1901.00238

[37] D. Rákos, "Programmable vertex pulling," in *OpenGL Insights*, P. Cozzi and C. Riccio, Eds. CRC Press, July 2012, pp. 293–301, http://www.openglinsights.com/.

[38] H. Wu, S. Gao, R. Wang, and J. Chen, "Fuzzy clustering based pseudo-swept volume decomposition for hexahedral meshing," *Computer-Aided Design*, vol. 96, pp. 42 – 58, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010448517301616

[39] J. Wu, N. Aage, R. Westermann, and O. Sigmund, "Infill optimization for additive manufacturing—approaching bone-like porous structures," *IEEE transactions on visualization and computer graphics*, vol. 24, no. 2, pp. 1127–1140, 2017.

[40] X. Gao, H. Shen, and D. Panozzo, "Feature preserving octree-based hexahedral meshing," *Computer Graphics Forum*, vol. 38, no. 5, pp. 135–149, 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13795

[41] M. Livesu, N. Pietroni, E. Puppo, A. Sheffer, and P. Cignoni, "Loopycuts: Practical feature-preserving block decomposition for strongly hex-dominant meshing," *ACM Transactions on Graphics*, vol. 39, no. 4, 2020.

[42] M. Livesu, N. Vining, A. Sheffer, J. Gregson, and R. Scateni, "Polycut: Monotone graph-cuts for polycube base-complex construction," *ACM Trans. Graph.*, vol. 32, no. 6, Nov. 2013. [Online]. Available: https://doi.org/10.1145/2508363.2508388


**Christoph Neuhauser** is a PhD candidate at the Computer Graphics and Visualization Group at the Technical University of Munich (TUM). He received his Bachelor's and Master's degrees in computer science from TUM in 2019 and 2020. Major interests in research comprise scientific visualization and real-time rendering.

**Junpeng Wang** is a PhD candidate in the Computer Graphics and Visualization Group at Technical University of Munich, Germany. He received his Bachelor and Master's degrees in Aerospace Science and Technology in 2015 and 2018, respectively, both from Northwestern Polytechnical University, China. Currently, his research is focused on tensor field visualization and numerical simulation for solid mechanics.

**Rüdiger Westermann** studied computer science at the Technical University Darmstadt and received his Ph.D. in computer science from the University of Dortmund, both in Germany. In 2002, he was appointed the chair of Computer Graphics and Visualization at TUM. His research interests comprise scalable data visualization and simulation algorithms, GPU computing, real-time rendering of large data, and uncertainty visualization.