HexaLab.net: an online viewer for hexahedral meshes

Matteo Bracci[1,*], Marco Tarini[2,*], Nico Pietroni[3], Marco Livesu[4], Paolo Cignoni[1]

Abstract

We introduce HexaLab: a WebGL application for real time visualization, exploration and assessment of hexahedral meshes. HexaLab can be used by simply opening www.hexalab.net. Our visualization tool targets both users and scholars. Practitioners who employ hexmeshes for Finite Element Analysis, can readily check mesh quality and assess its usability for simulation. Researchers involved in mesh generation may use HexaLab to perform a detailed analysis of the mesh structure, isolating weak points and testing new solutions to improve on the state of the art and generate high quality images. To this end, we support a wide variety of visualization and volume inspection tools. Our system offers also immediate access to a repository containing all the publicly available meshes produced with the most recent techniques for hexmesh generation. We believe HexaLab, providing a common tool for visualizing, assessing and distributing results, will push forward the recent strive for replicability in our scientific community.

Keywords: hexahedral mesh, hex-mesh, 3D visualization, volumetric remeshing, benchmarking, 3D web application

1. Introduction

Hex-meshes, i.e. volumetric meshes composed of hexahedral cells, are one the of most used 3D representations for numerical simulation, most notably by Finite Element Analysis (FEA). Application domains include structural mechanics, heat, electricity transfer problems and simulation of other physical phenomena. In order to be usable in a simulation, a hex-mesh must fulfill a number of requirements, both hard and soft in nature. In other terms, it must have a sufficiently high "quality".

The construction of simulation grade hex-meshes for a given shape is a long standing, extremely arduous problem, which has continuously attracted interest from industry and fuels a constant (and still ongoing) research effort by more than one scientific community (Sec. 2.1). The faced tasks are cast, for example, as the automatic and reliable construction of a hex-mesh of an object given its surface (hex-meshing); the "clean up" of a given hex-mesh form pathological configurations that prevent usability (untangling); or the conversion of a tetrahedral mesh into a hex-mesh (hex-remeshing).

We introduce *HexaLab*, an Open-Source software tool designed to help researchers and practitioners. HexaLab is both an advanced hex-meshes visualizer, and a portal to an online database of results produced with existing techniques. It is provided as a Web Application, and can be used by simply connecting to `www.hexalab.net`.

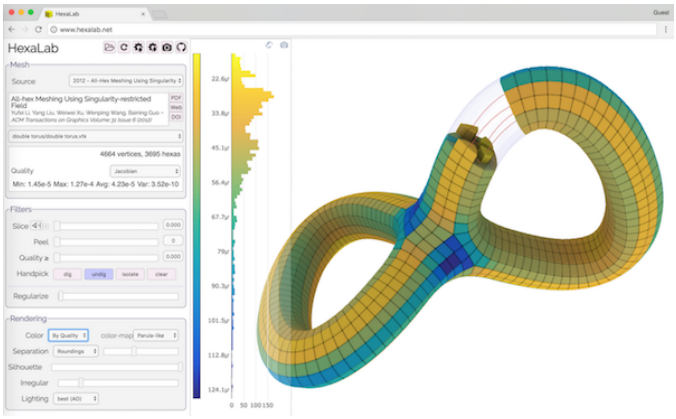


Figure 1: A screen-shot of HexaLab running inside Chrome under macOS. Model courtesy of [1].

1.1. Intended usages

Visualization: the main function of HexaLab is to serve as an advanced 3D visualizer for hex-meshes, for the purpose of providing a visual insight on the inspected models, and thus, indirectly, on the algorithms that produced them. The visualization of a hex-mesh is complicated by the resolution of the model and, even more so, by the presence of non-trivial 3D *internal* structures. HexaLab faces this task by offering a set of interactively controlled tools (cutting planes, etc), a shape-revealing realistic lighting (drastically improving the readability of images), novel modalities specifically designed to better communicate the shape of the cells, as well as colormaps and spatial glyphs, to reveal the quality, the topological characteristics, and so on.

Assessment: for the same purpose, HexaLab can be used

*Joint first author
[1]CNR ISTI, Pisa, Italy
[2]Università degli studi di Milano, Italy
[3]University of Technology, Sydney, Australia
[4]CNR IMATI, Genoa, Italy

to assess the quality of the inspected models, by providing interactive and automated techniques to perform numerical measurements and plotting histograms for the inspected model. A wide array of established measures are included.

Presentation: although they are produced in real-time in the context of an interactive application, images produced by HexaLab have publication-grade quality and are intended to be used for presentation and dissemination, e.g. in scientific publications. The high lighting quality helps to making 3D rendering intelligible even as static (e.g. printed) images. This function relieves researchers from the tedious task of implementing *ad-hoc* visualization method for the presentation of their results; also, such visualizations are often comparatively less effective, and are inhomogeneous across works from different authors.

Comparisons: HexaLab also implements mechanisms to ease comparison between hex-meshes, e.g. between results of competing methods over the same input, or, between "before" and "after" datasets of a hex-mesh optimization. All visualization settings can be easily shared between visualization sessions. This applies to sessions being run at the same time, for interactive side-to-side comparisons, and just as well to sessions executed much later in time. The visualization settings are also recorded in all produced images as metadata, to allow reproducibility of these images, and, thus, direct comparisons with new results in the future. Potentially, comparison is also indirectly fostered, even by results not directly dealing with each other, by their adoption of a shared visualization style.

Batch processing: in recent years, the number of results typically used to evaluate remeshing algorithms increased. For this reason, HexaLab can be used to read a collection of models, and produce images and measurements for them in one go.

Benchmarking: lastly, but perhaps most crucially, HexaLab is also an easily accessible portal to a new online repository of hex-meshes, which is already fairly complete, and, we believe, is destined to grow over time with the continued usage of the tool. Direct contributions of new datasets from authors of new methods are made simple and, in our intentions, encouraged (among other things) by the resulting visibility offered by HexaLab to any such works. The original source of any work in the repository is reported to all future users.

2. Background and Related Work

2.1. Hex-Mesh Generation and Processing

Hex-meshes are often preferred to other polyhedral based representations, such as tetrahedral meshes, because arguably they offer advantages in terms of numerical accuracy [3] or, equivalently, simulation speed for the same level of accuracy (due to the smaller number of degrees of freedom). Hex-meshes generation, however, is notoriously difficult.

In 1998 Owen [4] pointed out that hex-meshing techniques were not robust enough to be able to scale on complex shapes. Two years later, Blaker [5] defined hexmesh generation as the *holy grail* of meshing research. In the last decade, there has been a constant improvement in hex-meshing algorithms, mainly tied to advancements in volumetric parameterization techniques; yet, more research is clearly needed before fully satisfactory solutions are reached. Here we briefly summarize only a few of the most recent advances, and point the reader to [4] for a survey on classical methods developed during 90s. It is our opinion that availability of powerful tools to visualize, inspect, compare and analyze hex-meshes will help promoting further advance in the state of the art.

Grid based approaches [6] subdivide the volume using a regular grid. Elements are aligned to the global axes, and external vertices are snapped to the surface to better approximate the original shape. The most recent advances in the field regard the ability to project vertices on the surface while meeting per element quality bounds [7] and the introduction of templated schemes to turn hierarchical grids (octrees) into full hex-meshes [8]. The resulting meshes often expose a complex structure, which can be simplified in post-processing [9][10].

Advancing front approaches instead propagate element generation from the boundary toward the interior [11]. While these methods generate high quality elements close to the boundary surface, they tend to create badly shaped elements or leave empty voids in the interior, where different fronts meet. Alternatively, the front propagation may proceed *top to bottom*. In [12] the authors propagate the front following the integral curves of a harmonic function. Front splitting/merging at the saddle points of the guiding function is also supported.

One recent trend is based on *volumetric parameterizations*. The volume is first mapped on a space that admits trivial hexmeshing (e.g. via regular sampling); then the connectivity is generated in parametric space and projected in object space using the inverse map. Mesh is eventually cured [13] to remove imperfections. Common parametric spaces are cylinders [14, 15] and orthogonal polyhedra (or *polycubes* [16]). Approaches differ to each other on how the parametric space and the map are generated. High-quality polycubes should balance map distortion with corner count [17, 18][19], achieving a good singularity alignment [2]. As observed in [20], polycubes can be seen as a special case of field-aligned methods [21, 1, 22, 23]; the latter are more general, having internal singularities to improve the element shapes. In field-aligned methods, the mesh is generated by tracing integer iso-lines [24] of a volumetric parameterization aligned with a frame-field [25, 26]. The structure and quality of the mesh depend on the singularities and the smoothness of the guiding field, respectively. Unfortunately, not all the possible field singularities are compatible with hex-meshing, and a number of heuristics have been proposed to remove incompatible configurations [1, 22, 27]. The ro-

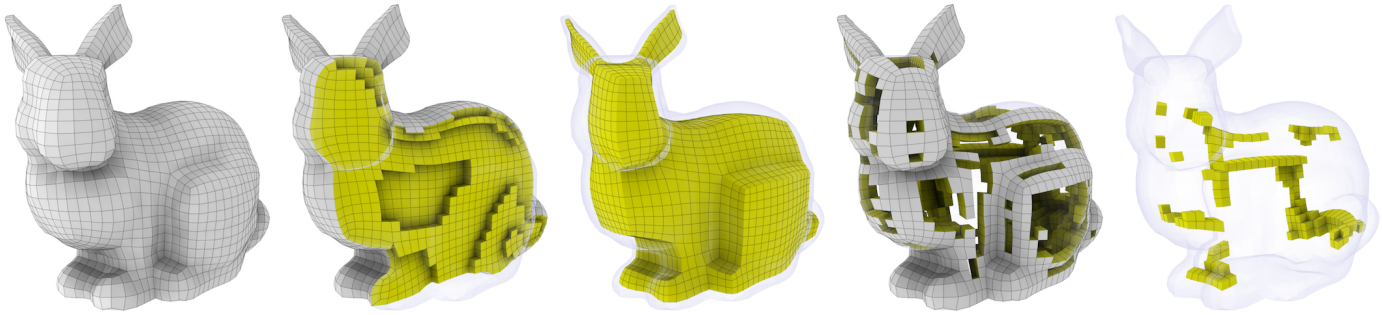


Figure 2: Examples of HexaLab filters applied to an hex-mesh to reveal its interior. From left to right: complete, unfiltered model; slicing-plane filter; peeling filter (two most external layers of cells removed); quality-thresholding filter (hexas with SJ < 0.96 are shown); and the combination of all these filters. "Bunny" model courtesy of [2].

bust generation of fields that admit a valid hex-mesh is an open problem. Alternatively, recent field-aligned methods targeting the construction of hex-dominant meshes [28, 29] can bypass this problem.

2.2. Hex-Mesh Software Tools

Alongside professional and well-established open tools for solid mesh generation and processing, such as Gmsh [30] and ParaView [31], there exists a variety of smaller tools being developed by researchers operating in the field and released to the community [32]. We give here a non comprehensive list of hex-mesh related software, considering both libraries and desktop applications.

A number of freely available libraries offer data structures [33][34][35] to import/export a hex-mesh and process it. These tools usually have a dedicated visualization front-end (e.g., Graphite [36] is the visual front-end of GeoGram [33]), but the functionalities they offer are more limited if compared to the ones we offer in HexaLab (e.g., no histograms, no advanced lighting, no advanced inspection tools for the mesh interior and structure).

Other software tools focus on mesh synthesis and processing, such as Hexotic (which implements [8] and is distributed by Distene within their MeshGems), LibHexEx [24] (which enables the extraction of a hexmesh starting from an integer grid map) or Mesquite [37] (which serves to maximize per element quality). These tools are orthogonal to us, as they focus on the processing of a hex-mesh but do not offer any visualization facility whatsoever.

Summarizing, our positioning is in-between professional tools and tools being developed and maintained by pure researchers: we offer high quality visualization and analysis tools which are better than the ones provided by the rest of the research community, at the expense of a simplified setup (we run on browser) which is much lighter than the one usually required by professional software.

2.3. 3D Online Visualization

Compared to desktop applications, web-applications are by design lightweight and have many several desirable characteristics, such as extreme portability (being based on a natively cross-platform technology), immediate availability to users (due to the absence of an installation phase), safety (due to browsers being protected environments), amenability to online and distributed applications (due to the inherent client/web-server setup), and easiness of deployment and maintenance (due to instant updates).

Traditionally, high-quality real-time 3D rendering of complex scenes has been considered a computationally demanding task, requiring GPU support, and therefore advanced visualization tools have been limited to desktop applications. In the last years, the arise of a widely supported web-based 3D API, WebGL by Khronos, fueled an increase of web-based 3D visualization tools.

Many of these tools target specific domains and are to use, embedding a small set of advanced specialized functionalities that are simple to understand. For example, in the field of molecular visualization, after the first Java-based approaches, like JMol[38], a number of small specialized visualization tools have been proposed [39, 40, 41]. Similarly, various web-based applications have been developed for volume visualizations tasks [42, 43, 44], confirming the appeal of specialized, lightweight visualization tools.

General-purpose tools to display collections of 3D models have also been one of the most common tasks for web-based visualization applications, such as Sketchfab or 3DHop [45], among many others; we refer the reader to [46] for a discussion of the possibilities and the issues of delivering 3D collections on the Web.

3. Overview

HexaLab can be used to visualize and assess the quality of hex-meshes either provided by the user in standard formats or downloaded from its own online integrated library of reference hex-meshes from recent literature (Sec. 4).

The visualization and exploration process is based on a very simple approach: in a trackball-controlled, real-time, high quality 3D rendering of the inspected mesh (Sec. 5) the user can interactively apply on it three different kinds of *exploration tools*:

Cell Filtering Tools: (Sec. 6) that allow users to hide portions of the hexahedral mesh so to expose internal structures which would otherwise be occluded. Criteria to hide cell can involve the use of a clipping plane, distance from mesh boundary, geometric quality, and handpicking of specific cells;

Quality Assessment Tools: (Sec. 7) that allow users to visually and numerically assess the geometric quality of the hexahedral cells, both individually (via color-coding), and as aggregated measures (through histograms and statistics), according to a number of standard quality measure (Sec. 7.1). Inverted (and concave) cells are treated and highlighted separately;

Global-structure Visualization Tools: (Sec. 8) that depict the configuration of irregular elements (especially edges) across the entire mesh under inspection, thus revealing its global topological structure.

HexaLab offers visualization status management (Sec. 9): at any moment, the current status of the interactive visualization can be readily captured, stored, and reused across visualization sessions. The status includes the setting of every active tool, the trackball-determined view-direction, any rendering parameters, the selected quality measure etc. This achieves reproducibility of any previously obtained image and allows visual comparisons, under the exact same conditions, of different datasets (e.g. for comparing alternative meshings of the same object).

While in this work we do not focus on interfaces, we strive to complete HexaLab with a reasonably intuitive Graphic User Interface (GUI). GUI elements will be described in the following sections, contextually with the mechanisms they control.

4. Online mesh repository

HexaLab doubles as an easily accessible, online repository of the publicly available hex-meshes produced by different techniques appeared in recent literature. Currently, the repository consists of a total of 272 hex-meshes, from 15 different papers published in the last 7 years: [47, 1, 19, 18, 48, 20, 2, 49, 15, 50, 51, 52, 14, 53, 54].

This collection of hex-meshes is stored, together with the sources, on the git repository of HexaLab; it is made available directly by the HexaLab GUI: the user can simply invoke the visualization of any stored hex-mesh, by selecting one source, and then one model from that source. The requested hex-mesh is then automatically downloaded from the repository, and presented to the user. Contextually, HexaLab fully reports the data source by providing the bibliographic reference, a link to the DOI, and when available, to the PDF and the web page of the referenced article.

The maintenance and updating of the repository, when new results will appear, or other authors will make their data available, will be done by relying on the well-known *pull-request* mechanism made available by the GIT distributed version control system. In this way, we offer to the research community a direct and simple way of proposing additions of results to the repository, which will in turn foster comparisons against further advancements.

5. Hexahedral Mesh Rendering

The core part of HexaLab consists in the visualization of the inspected hex-mesh as a solid object with a set of cells filtered out (see Sec. 6), so as to reveal the interior structure.

Typically, the visualization of a polyhedral volumetric mesh $\mathcal{M}_V$ renders just the boundary $\mathcal{M}_S$ (a polygonal surface), using a flat shading model and with mesh edges overdrawn as line segments (see e.g. [36]). In our case, $\mathcal{M}_V$ is a pure hexahedral mesh, and $\mathcal{M}_S$ is a pure quad mesh.

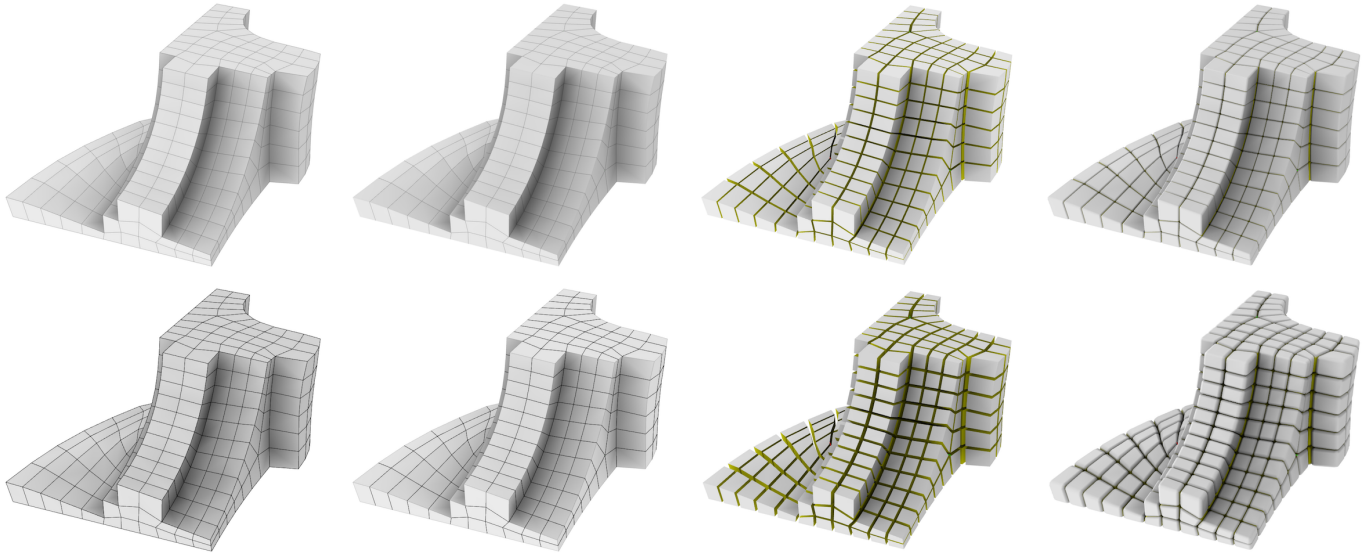
We observe that this common edge rendering choice has a significant shortcoming: it is impossible to distinguish how many different cells are incident to an edge. Indeed, the edges of $\mathcal{M}_S$ which separate difference cells of $\mathcal{M}_V$ and the edges which just separate different faces of the *same* cell of $\mathcal{M}_V$ are depicted in the same way.

A brute force approach to overcome this edge ambiguity problem appears in [29], and consists in drawing, instead of just the surface, the whole hex-mesh $\mathcal{M}_V$ with each cell slightly smaller. This approach, that technically exposes each face and therefore the local arrangement of cells near the boundary, does not scale well in terms of performances, due to the cubic explosion of primitives to be rendered at screen.

In HexaLab we propose three visualization modes to disambiguate the surfaces representing the volumetric mesh:

Darkness-coded edges: in the wireframe, we color the edges with thin, semi-transparent darker lines when they just separate two faces of the same cell. Opacity is higher for edges separating different cells, and progressively increased for any additional internal elements sharing such edge. In other words, the opacity of each edge on $\mathcal{M}_S$ is made proportional to the number of (non-hidden) cell elements in $\mathcal{M}_V$ sharing the boundary edge. This method is undemanding in terms of resources, and greatly helps disambiguating. On the downsides, it is not self-explicative, as the mapping between line opacity and number of elements is arbitrary (see Fig. 3, b).

Fissure mode: we separate adjacent cells by a small fissure so to reveal the local inner structure of the mesh. For the sake of rendering speed, fissures are limited to cells on the boundary of $\mathcal{M}_V$ and, as explained in Sec. 10.3.1, the rest of the mesh will remain occluded. Specifically, we only separate edges and faces of $\mathcal{M}_V$ which share at least one vertex on $\mathcal{M}_S$ (see Fig. 3, c).



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

Figure 3: (a) the traditional rendering of $\mathcal{M}_S$ to represent $\mathcal{M}_V$. (b) darkness-coded mode. (c) crack mode. (d) rounded mode. Bottom row: the same modes, for a different selection of darkness/width/radius. Model courtesy of [1].

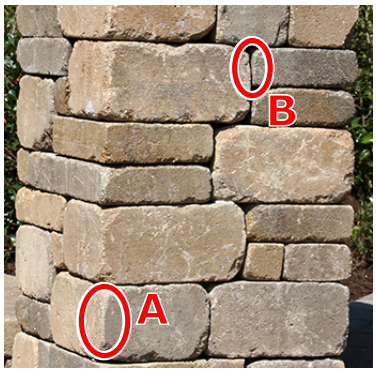


Figure 4: In this natural image of a stone brick structure, the edges that separate faces belonging to the same volumetric cell (e.g. in circle A) are immediately recognizable from edges that separate distinct volumetric cells (e.g. in circle B), although no line is explicitly added and cells are in full side-to-side contact. This inspires our Rounded mode for visualizing hex-meshes, which exploits a similar principle.

While this method helps solving the visual ambiguity we are addressing, it tends to clutter the screen and sometimes it is not readable.

Rounded mode: we observe that there are real world examples of cell arrangements (see Fig 4) where the structure is immediately recognizable. This inspired us to define an additional visualization technique where each cell is rounded around the edge. Technically, this leaves a small elongated gap around each edge of the mesh. Similarly to the above mode, we only render the surface of this gap in the immediate proximity of the surface. More details are found in Section 10.3. We found this method to produce the most readable images (see Fig. 3, d). On the other hand, this mode is a bit more resource demanding (the number of rendering primitives increases by an average factor of 12), and it can be argued that this adds geometric features (the rounded corners) which are non-existing in the input dataset.

Visualization modes are offered to the user as three alternatives, selectable via a combo-box. The three modes can be further customized by editing one single parameter, which determines respectively: an opacity multiplier for the Darkness-coded edges mode; the gap size for Fissure mode; and round radius for Rounded mode (Fig. 3, bottom). Since the meaning of these parameters is similar across all the modes, we avoid cluttering the interface and present the user a single slider to choose the parameter which is appropriate for the currently selected mode.

5.1. Lighting and Shading

It has been observed several times [55, 56] that a global lighting model is extremely helpful to facilitate a thorough comprehension of the shape of a 3D object. This is especially true when the depicted shapes are not necessarily familiar to the observer, as it is the case for hex-meshes under arbitrary filters. The problem is further exacerbated for static images (for example, in a scientific article) where the observer cannot rely on interactivity to disambiguate. For this reason, we consider rendering hex-meshes with only a local direct illumination inadequate, and we employ a view-independent Ambient Occlusion (AO) term [57, 58, 59] that can be pre-computed and stored at vertices of $\mathcal{M}_S$ (Fig. 5). The computation of the AO terms happens automatically in background whenever is needed, and takes only a few seconds to complete (less than 5s for the most complex model in the database,

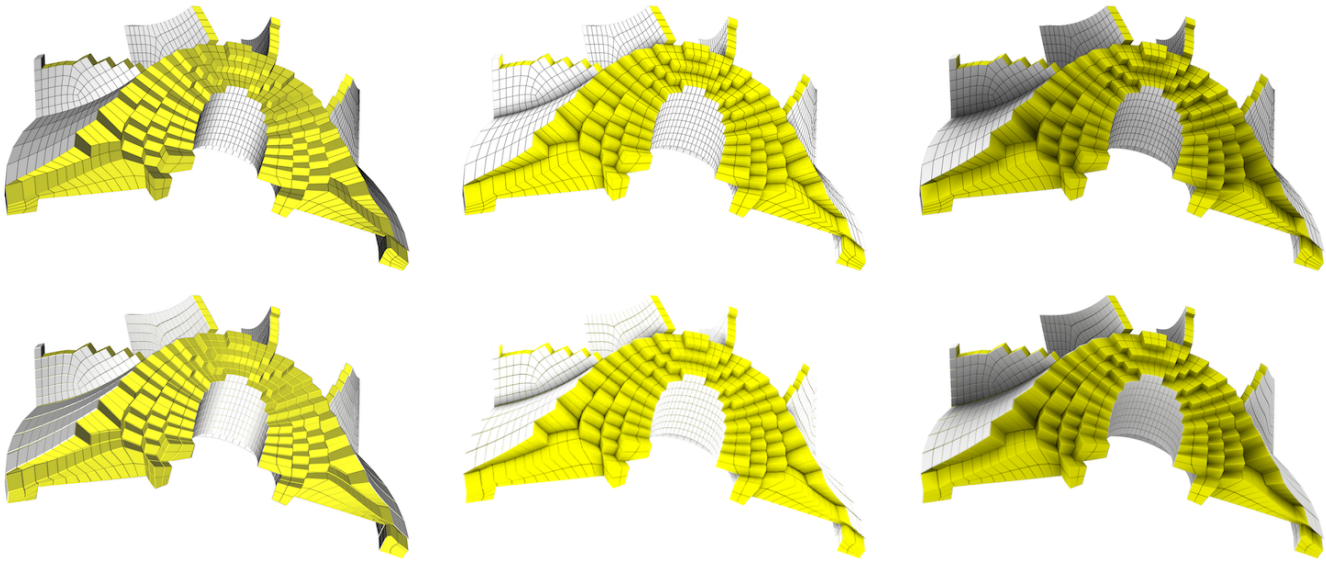


Figure 5: The lighting modes offered by HexaLab. From left to right: direct illumination only, global illumination approximated with Screen Space Ambient Occlusion (SSAO), and with Object Space Ambient Occlusion (OSAO), which is HexaLab default. Global illumination schemas are more demanding, but are more effective at convening the 3D space, especially OSAO. Bottom: the improvement is further enhanced when "rounded mode" is used to separate elements. Model courtesy of [1].

on a 2012 MacBook Pro). For the sake of interactivity, in the short time prior completion a screen-space approximation is used [60, 61, 62] as a temporary fall-back mode. This allows users, for example, to interactively sweep the slicing plane through the volume while still seeing a comprehensible representation.

When the color is not used to map element quality (see Sec. 7), HexaLab defaults a simple yellow–white color scheme to differentiate faces of $\mathcal{M}_S$ found on original mesh boundary of the mesh (prior to any filtering operation) from interior faces.

6. Cell filters

HexaLab provides four ways to filter away elements. Filters all work in a consistent way, that is, by flagging certain cells as hidden, and temporarily removing them from the visualization. Filters differ to each other only on the way they flag elements, and can be used either alone or through any combination of them (Fig. 2).

Slicing plane: any cell whose barycenter falls behind a user specified "slicing plane" is filtered out from the view. In other words, the mesh is intersected with a half-space bounded by the slicing plane;

Peeling: any cell with a hop distance from the original mesh boundary smaller than a user-specified "minimal depth" value is filtered away (Fig. 6); the hop distance is computed using face-to-face adjacency. In other words, a number equal to the "minimal depth" of successive peels are removed in succession from the original mesh, A peel is defined as the union of the cell elements with an exposed face (Fig. 6);

Quality Thresholding: any cell with a quality not worse than a user-specified "quality" is removed. This is useful to isolate and highlight the problematic regions, hiding the good ones;

Manual Selection: any selection obtained with the previous filters can be manually tweaked by using two operations: addition and removal of manually selected elements. These operations are triggered by pointing on a quad face f on the boundary of the currently displayed mesh: the "dig" tool hides the non-hidden cell that shares f; the "undig" tool reveals the hidden cell that shares f (if it exists). In both cases, the selected cell is uniquely identified by f. Finally, the "isolate" tool hides all cells except the selected one (neighboring cells can then be progressively added with the "undig" tools); this is intended as a way to help users discerning the structure of intricate configurations. An example is shown in Fig. 7, where, by undigging a few hidden cells, it is possible to easily understand the mesh configuration.

6.1. Filter regularization

The user can elect to automatically polish the selection by increasing the regularity of the resulting boundary (i.e. the quad mesh) to some extent. We implement this with a morphological approach. Specifically, the set of filtered elements is first dilated n times and then eroded n times, for a "strength" integer parameters n (ranging between 0 and 5). An erosion consists in the removal from the filtered set of any cell with a vertex on the current boundary, and vice versa a dilation consists in the addition. When this

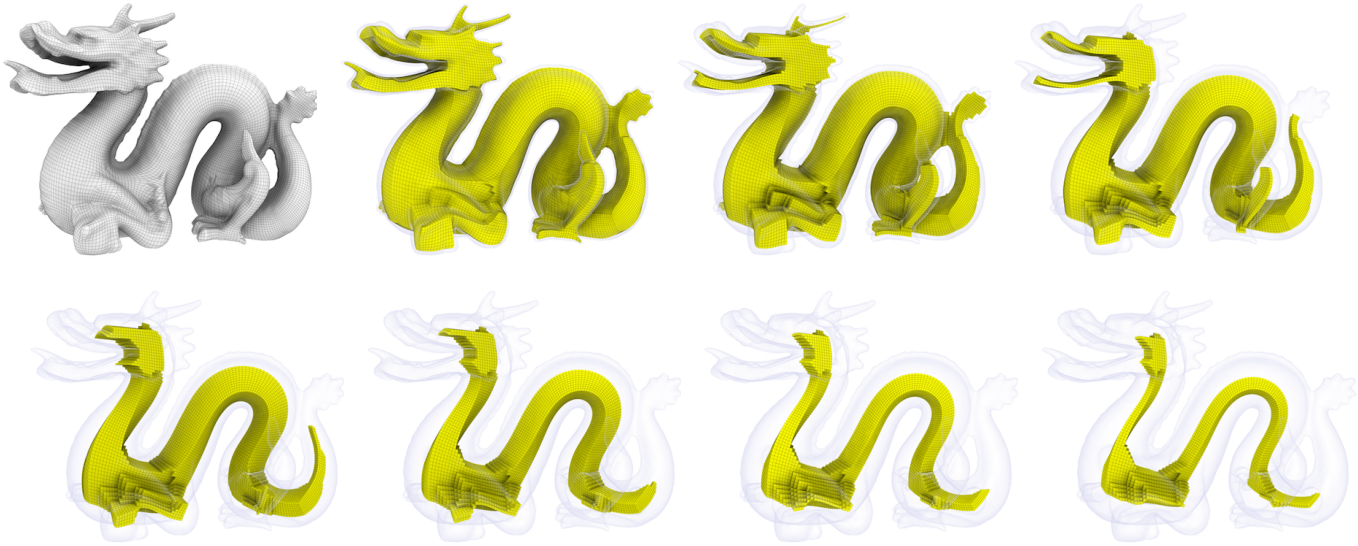


Figure 6: Several layers of peeling reveal hexahedral structure of the dragon follows the overall shape of the dragon. Model courtesy of [18].

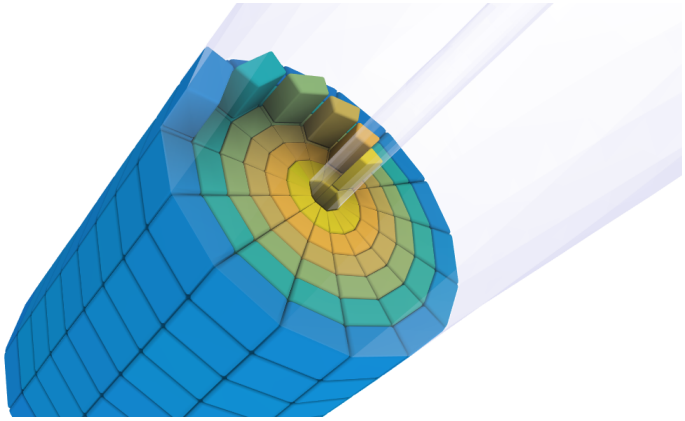


Figure 7: Sometimes, revealing (*undigging*) a few hidden cells produce an image which communicates the structure of the meshing in a very intuitive way. Model courtesy of [14].

option is activated, the selection is regularized after every change of the slicing plane or the peeling tool (the other filters are unaffected). A regularized selection can help clarifying the mesh topology by revealing the structure of internal planes (see Fig. 8).

6.2. Displaying filtered elements

The elements which are filtered out are (almost entirely) omitted from the rendering, in order not to clutter the view and not impact the readability of the currently exposed internal part. Optionally, they can be displayed as a pale silhouette, either uniformly colored or with a very light shading, for the purpose of providing a visual hint of the spatial context, in the original mesh, of the currently visible parts (see Fig. 9).

6.3. GUI and customization

The user controls the visibility of the filtered elements with one slider (labeled "silhouette"). The leftmost position completely hides filtered elements, whereas sliding to the right progressively shows the pale silhouette. By default, the slider is at the rightmost position.

The *peeling* and *quality thresholding* are controlled with a single slider determining the minimal depth and the maximal quality respectively. The *slicing plane* tool requires the user to identify the desired plane. Providing an intuitive interface for the selection of an arbitrary plane is not straightforward; we bypass this problem by leveraging the trackball, and simply providing a command (activated by a "set plane" button on the GUI) which uses the current view direction as the normal of the slicing plane (optionally, shift clicking this button round this vector to the closest axis-aligned direction). The facing of the plane is selected so that the cut surface faces toward the user. The offset of the plane then is controlled with a slider, analogous to the other two filters. To increase simplicity of use, the full-scale values of the three sliders are normalized, at both ends, so that the left extreme always means a null filter (all cells are visible), and the right extreme to mean a complete filter (all cells are hidden). To this end, HexaLab silently computes, and keeps up-to-date, data for the current mesh like maximal and minimal extension along the slicing plane direction, depth of most internal element, and quality range over all the elements.

A potentially useful operation that the user might want to perform is to invert the current orientation of the slicing plane, so that the filtered portion of the mesh is reversed, and the region of interest can be investigated "from the other side". This functionality can be accessed with a designated button, but can also be triggered by the set-plane button (if, upon activating it, the current view direction is detected to be close to opposite to the current slicing

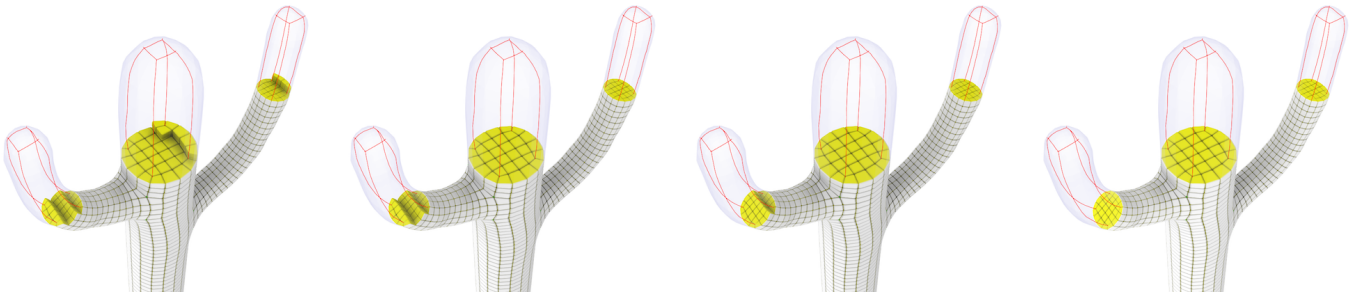


Figure 8: An example of regularization filter in action, with strengths ranging from 0 (left) to 4 (right): the boundary of the set of the cells removed by the slicing plane becomes increasingly clean. Model courtesy of [15].

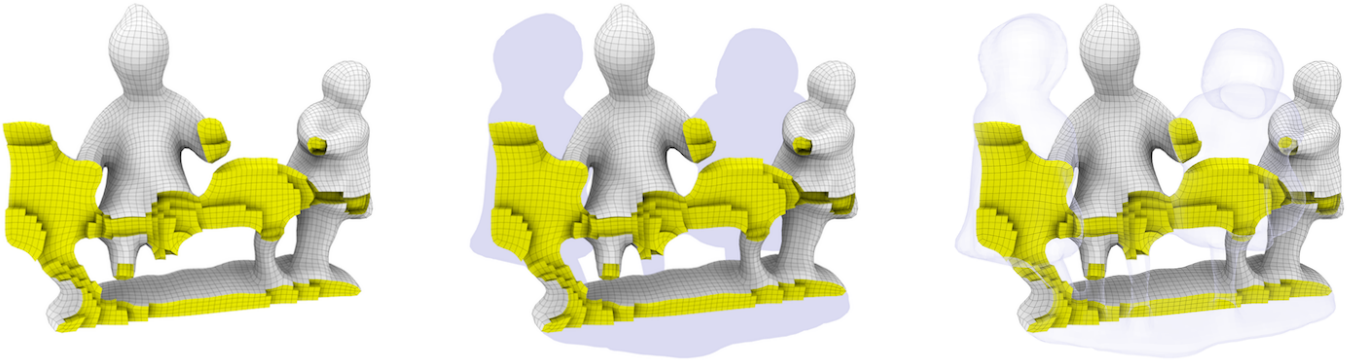


Figure 9: Left: no silhouette. Middle: flat silhouette. Right: semitransparent silhouette. Model courtesy of [49].

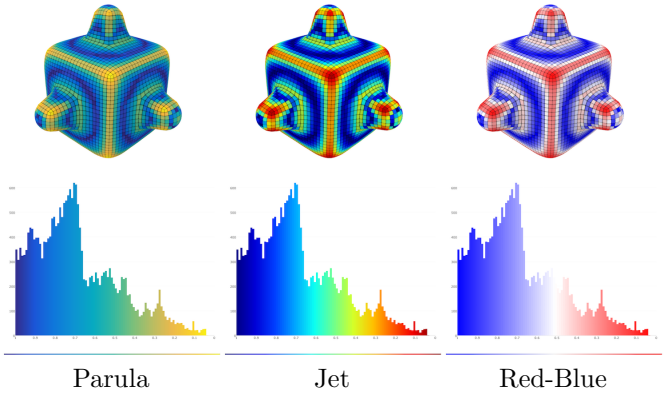


Figure 10: Different color mappings on hex quality (top), and associated histograms (bottom). Model courtesy of [2].

plane direction – within a tolerance of 20 degrees – then the slicing plane direction is exactly inverted).

7. Cell Quality Visualization

The quality of individual cells is important in many applications, like Finite Element Analysis where a single badly-shaped element can impair the entire simulation.

HexaLab allows users to color hex-cells according to their quality using one of three color-maps, shown in Fig. 10, bottom, that have been chosen for their readability and wide circulation. The quality coloring works in addition to the filtering mechanism which isolates badly shaped elements by hiding all elements which are better than a prescribed threshold measure (see Sec. 6 and Figure 2, right).

We also display a histogram to show the distribution of the quality values among all the elements (Fig. 10, bottom right). The histogram, which consists of either vertical or horizontal bars (users' choice), can be saved as a subimage, it is color-coded using the currently selected color-map, and doubles as a labelled legend for the current color-coding on the 3D rendering.

7.1. Supported quality metrics

There is a wide spectrum of different metrics that can be useful to assess the quality and other characteristics of a hex-mesh. Many of these metrics refer to the deviance from the *ideal* shape, that is, a perfect cube having planar faces, orthogonal angles and all edges with equal length. Broadly speaking, the larger the deviation from the ideal shape, the more inaccurate the results produced by a FEA simulations can be expected. In particular, elements with nearly zero volume (i.e., *degenerate*) or negative volume (i.e., *inverted*) may introduce a significant error, or even preclude the entire simulation [48]. For a discussion of these metrics, the reader is referred to the recent interesting study [64].

HexaLab supports all the quality metrics reported in Verdict [63], that are the most widely adopted library in

Metric	ID	Full range	Acceptable range	Value for unit cube	$\Phi(q)$	$\Phi^{-1}(q)$
Diagonal	DIA	$[0, 1]$	$[0.65, 1]$	1	q	q
Distortion	DIS	$(-\infty, \infty)$	$[0.5, 1]$	1	$\frac{q-q_{min}}{q_{max}-q_{min}}$	$q_{min} + q(q_{max} - q_{min})$
Edge Ratio	ER	$[1, \infty)$	—	1	$\frac{q_{max}-q}{q_{max}-1}$	$1 + (1 - q)(q_{max} - 1)$
Jacobian	J	$(-\infty, \infty)$	$[0, \infty)$	1	$\frac{q-q_{min}}{q_{max}-q_{min}}$	$q_{min} + q(q_{max} - q_{min})$
Maximum Edge Ratio	MER	$[1, \infty)$	$[1, 1.3]$	1	$\frac{q_{max}-q}{q_{max}-1}$	$1 + (1 - q)(q_{max} - 1)$
Maximum Asp. Frobenius	MAAF	$[1, \infty)$	$[1, 3]$	1	$\frac{q_{max}-q}{q_{max}-1}$	$1 + (1 - q)(q_{max} - 1)$
Mean Asp. Frobenius	MEAF	$[1, \infty)$	$[1, 3]$	1	$\frac{q_{max}-q}{q_{max}-1}$	$1 + (1 - q)(q_{max} - 1)$
Oddy	ODD	$[0, \infty)$	$[0, 0.5]$	0	$\frac{q_{max}-q}{q_{max}}$	$(1 - q)\, q_{max}$
Relative Size Squared	RSS	$[0, 1]$	$[0.5, 1]$	—	q	q
Scaled Jacobian	SJ	$[-1, 1]$	$[0.5, 1]$	1	$\max(q, 0)$	q
Shape	SHA	$[0, 1]$	$[0.3, 1]$	1	q	q
Shape and Size	SHAS	$[0, 1]$	$[0.2, 1]$	—	q	q
Shear	SHE	$[0, 1]$	$[0.3, 1]$	1	q	q
Shear and Size	SHES	$[0, 1]$	$[0.2, 1]$	—	q	q
Skew	SKE	$[0, \infty)$	$[0, 0.5]$	0	$\frac{q_{max}-q}{q_{max}}$	$(1 - q)\, q_{max}$
Stretch	STR	$[0, \infty)$	$[0.25, 1]$	1	$\frac{q}{q_{max}}$	$q\, q_{max}$
Taper	TAP	$[0, \infty)$	$[0, 0.5]$	0	$\frac{q_{max}-q}{q_{max}}$	$(1 - q)\, q_{max}$
Volume (signed)	VOL	$(-\infty, \infty)$	$[0, \infty)$	1	$\frac{q-q_{min}}{q_{max}-q_{min}}$	$q_{min} + q(q_{max} - q_{min})$

Table 1: List of per-element metrics supported in HexaLab. To offer a consistent color-coded quality visualization, we map each metric in the normalized interval $[0, 1]$, where 0 corresponds to the worst quality and 1 to the best quality. The functions we use to move from the native range to the normalized range (Φ) and vice-versa (Φ^{-1}) are shown in the two rightmost columns of the table. For unbounded metrics, q_{max} and q_{min} refer to the highest and lowest quality values measured in the mesh. For details on the computation of each metric, the reader is referred to [63].

the field for the evaluation of finite elements. We report them in Table 1.

7.2. Metric normalization

The supported metrics differ widely in range and behavior; for example the range is bounded for some measure (for example, the *Scaled Jacobian*) and unbounded for others (for example, cell *volume*); optimal value can be the lowest or the highest value in the range.

In spite of this heterogeneity, in order to manage the metrics in a consistent yet intuitive way, HexaLab *internally* stores per-cell element values in a normalized interval $[0, 1]$, with the convention (where appropriate) that the value 0 always corresponds to the worst quality, and the value 1 to the best. To this end, we have designed, for each metric i, an ad-hoc function Φ_i which maps the original range of that metric into the normalized interval. For metrics with unbounded ranges, Φ_i is also a function of the maximum and minimum values found in the current mesh (which are updated at load time). All these mapping functions are reported in Table 1.

Thanks to this normalization, HexaLab will consistently color the worst elements (for example, as *red* under "Jet" color map), will consistently isolate the worst elements with the quality filter, and will consistently account for the worst elements in the right (or bottom) end of the histograms, regardless of the currently selected measure. The normalized values, however, are only used internally and never directly exposed to the user: HexaLab always uses the original values of the current metric on all histogram labels, legends, and GUI elements (e.g. the box containing the threshold value of the quality filters). This is well-defined as all our functions Φ_i are invertible (technically, with the exception below).

The Scaled Jacobian is one of the most widely adopted quality measures for hexmeshes, and is the default measure in HexaLab. For this measure, which is defined in the range $[-1, 1]$, Φ_i simply clamps negative values to 0. We consider this mapping more useful than a linear mapping from $[-1, 1]$ into $[0, 1]$. First of all, Φ_i correctly assigns a very poor quality to cells with a Scaled Jacobian value close to zero. Also, cells with inverted (or degenerate) corners, which can be argued to be equally invalid in

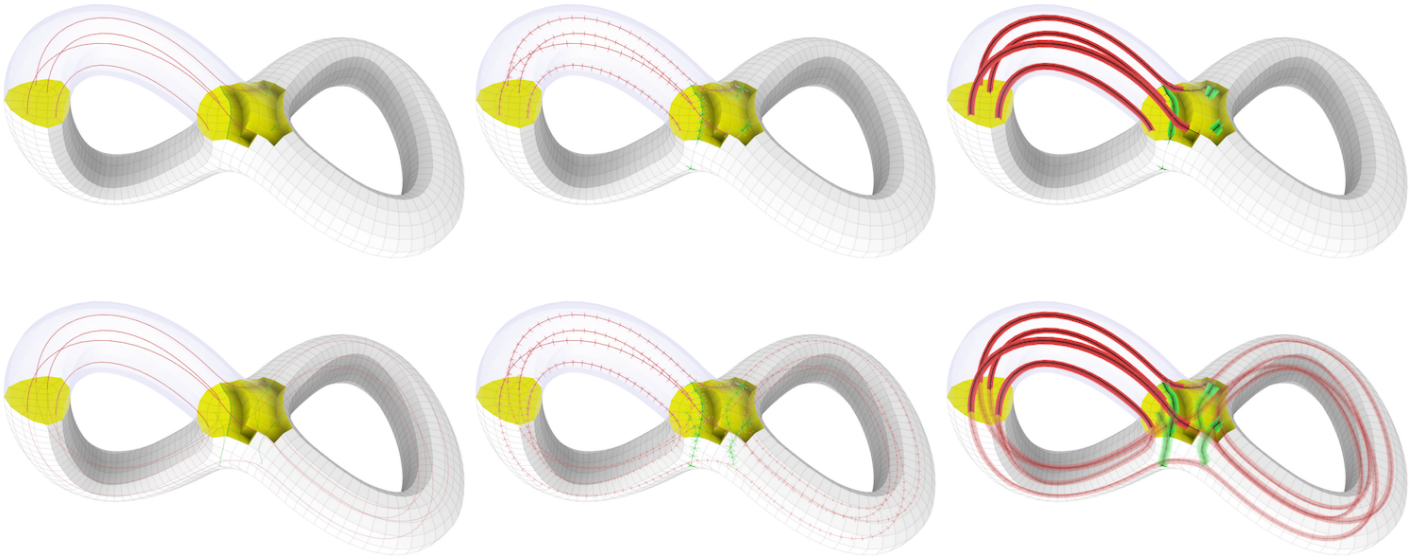


Figure 11: The sequence of visualization modes for the irregular structures. In increasing order of amount of information provided: "wire" mode (default), i.e. as lines colored as a function of edge arity; "barbed wires" mode, i.e. with adding all edges stemming out from the irregular edges; "paper" mode, where incident faces are also partially shown. In modalities variants shown on bottom, HexaLab reveals in transparency elements which would otherwise be occluded by currently un-filtered hexas. Model courtesy of [1].

hexmeshes, are clustered in one histogram bin (zero); consequently, the first bin of the histogram counts inverted (and degenerate) cells, and pushing the quality filter all the way to the right hides every cell but inverted (and degenerate) ones.

8. Mesh structure Visualization

In a structured hex-mesh, internal vertices are shared by eight cells, and internal edges by four cells; boundary edges (edges lying on the boundary of a mesh) are shared by two cells, and boundary vertices by four. Elements (vertices or edges) with a mismatching number of adjacent cells are termed *irregular*. Irregular edges are necessarily organized in strips which traverse the mesh from an irregular vertex to another (either on the surface or in the interior of the mesh).

Similarly to irregular vertices on quad meshing, irregular elements on hex-meshes characterize how cells are globally organized. For example, a particular class of hex-meshes, called *generalized* polycubes [65], have no internal irregular edges. Internal irregular elements, often referred to as "meshing singularities", have been closely investigated for their direct relationship with the subdivision of hex-meshes into fully structured sub-blocks [66, 9]. In general, the number, disposition and connection of irregular edges are linked to important properties of the hex-mesh.

HexaLab offers a variety of visualization modalities to depict irregular elements. In each variant, different primitives are used to convey the presence of irregular edges, and (some of them) associated info like their valence. All the modalities are illustrated in Fig. 11.

We offer a series of visualization modalities for irregular elements which range from lightly hinted to most informative (but also potentially cluttering and confusing the rest of the visualization). In the most informative modality, the irregular structure is visible in semi-transparency through the cell elements, whereas with the lighter modes it is only exposed in areas where cells are hidden by filters. The user selects one modality through a slider having as many discrete steps as the number of possible choices. The leftmost step disables the irregular-element visualization entirely, while the rightmost step corresponds to the most informative modality.

9. Status snapshots

The current "status" of the application, which includes current settings for all visualization tools and camera parameters, is unambiguously described in textual format, as JSON object. This includes, for example, the orientation and position of the slicing plane, the view direction, the used colors, and so on. The main intended use is to ease direct comparisons between different hex-meshes (see Fig. 12), and it is also the base for a number of additional mechanisms:

Manual status copy-pasting: the user can copy the status to and from the system clipboard (via the copy and paste keyboard shortcuts), and thus manually transfer the visualization settings across simultaneously open instances of HexaLab (or across consecutive visualization sessions, storing them for further reference);

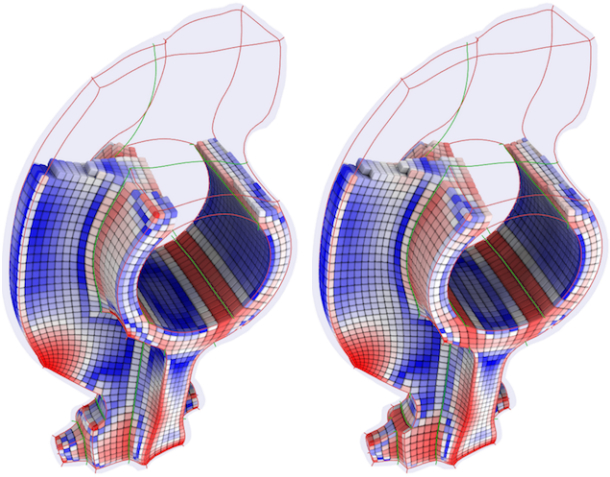


Figure 12: Two alternative meshings for the Rockerarm model (in this instance, the input and output of the technique [2]) visually compared in HexaLab by using with the same *status*. The status has been copy-pasted across visualization sessions.

Image reproducibility: the status is automatically attached as meta-data to every produced image (via a snapshot tool), for future reference. Dragging a snapshot of a previous session into HexaLab, meta-data will be read and the status of the web application automatically updated so as to reproduce the same visual contained in the image. All the images contained in this paper have been produced with this tool. Exporting images from the original version of the manuscript and dragging them into HexaLab will allow readers to see the same images in their browsers;

Manual parameter tweaking: as the setup is saved on a text file, the user can manually edit every parameter. While not intended as the main way to interact with HexaLab, this allows for full control (e.g. to pinpoint the orientation of the cutting plane, to select certain cells by numeric ID, to choose colors). Secondarily, it partially exempts the graphic interface from the need to provide full access to each parameter, allowing us to simplify it;

10. Implementation

The software architecture of HexaLab loosely follows the Model-View-Controller design pattern [67]. Two modules are integrated in a single web application: a back-end module that deals with mesh input/output, storage, and manipulation, and a front-end module for the rendering and the GUI.

10.1. Internal hex-mesh representation

Different representations for polyhedral meshes offer several tradeoffs between navigation efficiency and memory footprint. Our requirements include the necessity to keep the latency time low when the current filtering changes and a new quad-mesh boundary of visible elements needs to be produced. We rely on assumption that the mesh connectivity is fixed, and that the mesh resolution is in the range of around 10^5 cell elements.

We opted to store the hex-mesh $\mathcal{M}_V$ as a *generalized map*. One reason is that homogeneous array sizes simplify algorithms; also, this makes HexaLab amenable to include support of non hexahedral polyhedra in future expansions.

Generalized maps are a slightly more general but less compact version of a combinatorial map; HexaLab uses its own implementation, although other implementations are available [68, 69]. They are formalized in [70, 71], and here we only briefly recap them. The mesh connectivity is stored as a collection of "darts", which are a generalization of the half-edge structure commonly used for polygonal meshes. A dart represents a topological location on the mesh, and consists of a collection of four indices to one 3D, 2D, 1D, and 0D element of the mesh (i.e. a cell, a face, an edge, and a vertex). Additionally, each dart stores a set of four indices (called "involutions") pointing to the dart reached if any of its four elements is swapped (invalid indices are stored at darts at mesh boundaries). Involutions provide an efficient mean to navigate over the mesh. We precompute and store darts immediately after mesh import, leveraging standard associative container structures (such as black-red trees). Finally, we store attributes at mesh elements, such as (x, y, z) positions at vertices, normal vectors at faces, and "filtered-status" (a Boolean variable) at cells.

10.2. Rendering algorithms

The extracted quad-mesh $\mathcal{M}_S$ is stored as an indexed triangular mesh, duplicating vertices when we have to represent normal or color discontinuities along edges. Quads are simply split into triangle pairs along an arbitrary diagonal (although rendering methods which bypass the need for this arbitrary split have been proposed, [72]).

By default, Ambient Occlusion (AO) is the only illumination component used in HexaLab and substitutes direct illumination entirely. Similarly to [56], AO terms are iteratively computed in object space, by superimposing a sequence of shadow-maps, one for each probe light direction: in our case, we accumulate unblocked light, weighted by the *Cosine law*, directly at vertices of $\mathcal{M}_S$; note that this potentially produces discontinuity of ambient occlusion factors at normal discontinuities, as expected. We use a set of 1024 probe lights directions, which sample the unit sphere in an approximately uniform way, and are randomly constructed approximating a blue noise distribution. All renderings, including of the shadow-map, and the accumulate light on the per-vertex AO terms, are performed in GPU, leveraging WebGL.

AO terms are updated in background, accumulating contributions from light probes without compromising interactivity. After any user-triggered update of $\mathcal{M}_S$, buffered AO terms are discarded, and their computation

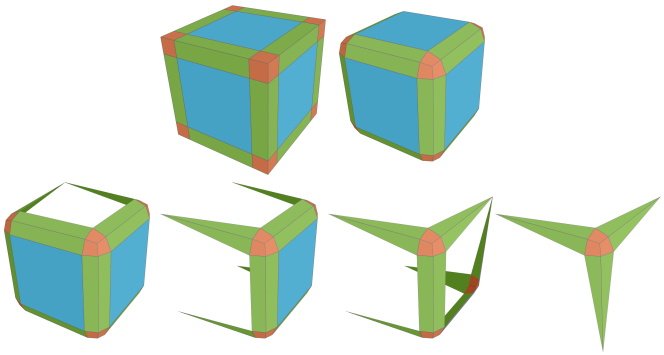


Figure 13: Boundary polygons produced for a hexacell (to be rendered and lit). Top: topological subdivision and geometrical displacement. Bottom: examples of the produced polygons for four different configurations of six, four, three, and one "exposed" vertices.

restarted. We wait that at least six light probes are accumulated in the AO terms before using them in any rendering; before then, we fall back screen-space AO approximation [60, 61, 62].

10.3. Polygonal mesh extraction

After each user filter operation, the hidden cells are removed, then HexaLab dynamically extracts a polygonal surface-mesh $\mathcal{M}_S$ to be globally lit and rendered from $\mathcal{M}_V$.

In *darkness-coded edges* mode, $\mathcal{M}_S$ is simply composed of the quad faces of $\mathcal{M}_V$ separating one filtered and one unfiltered cell of $\mathcal{M}_V$. In *fissure* and *rounded* mode, visible cells undergo certain spatial deformations, affecting both the polygon connectivity and the geometry of $\mathcal{M}_S$, and requiring specialized algorithms to extract it. Specifically, in *fissure* mode, cells are shrunk, forming gaps around quads of $\mathcal{M}_V$; in *rounded* mode, cells edges are rounded, forming, as a side effect, small tubular gaps around edges of $\mathcal{M}_V$.

For the latter two modes, our mesh extraction algorithm consists of two steps: first we identify the vertices of $\mathcal{M}_V$ which lie on its boundary (given the current filtering), and we label them as *exposed*; next, we process any cell with at least one exposed vertex, and we add vertices and polygons to $\mathcal{M}_S$ according to the exposed status of the eight corner vertices of that cell, as detailed in Sec. 10.3.1 and 10.3.2.

In all modes, $\mathcal{M}_S$ is always a closed and geometrically watertight polygonal mesh. Its vertices are produced with positions, base color, and normals (used by the global lighting). In *darkness-coded edges* and *fissure* mode, we employ flat shading, so each quad of $\mathcal{M}_S$ indexes its own instances of the vertices (producing hard creases between all neighboring faces). In *rounded* mode, we employ smooth shading, and polygons of $\mathcal{M}_S$ belonging to the same cell index share vertices (so hard creases only appear *between* cells).

10.3.1. Mesh extraction in Fissure mode

We produce a quad for each cell face sharing at least one exposed vertex (producing all four $\mathcal{M}_S$ vertices for that face, irrespective of the "exposed" status of the corresponding $\mathcal{M}_V$ vertex). Exposed vertices only are then moved toward the barycenter of the cell, covering a small, user-selected percentage of the distance. This way, inter-cell gaps are formed, but only in the proximity of the boundary of the visible parts of $\mathcal{M}_V$; because the unexposed vertices are kept in their original positions, the gaps close going toward the internal portions of the hex-mesh, and $\mathcal{M}_S$ has no element at all in the more internal parts.

This introduces an approximation, as in reality every side of every cell of $\mathcal{M}_V$ would be exposed due to all fissures forming one connected empty space. The visual effect of the approximation is, however, extremely small, because it takes place in regions which are typically both occluded by more superficial elements and made dark by the global lighting. The approximation drastically improves performances.

Because vertices are moved toward the barycenter of the respective cell covering a fixed proportion of the distance, the schema automatically adapts to varying sizes of the cells (e.g. narrower gaps are formed around smaller cells).

10.3.2. Mesh extraction in Rounded mode

In this mode, we subdivide each side of the processed cell into 3×3 squared sub-faces; this produces four additional vertices at each cell face, and two additional vertices at each cell edge (see Fig. 13). The vertices on the edges and corners of the cell are displaced toward the barycenter of the respective edges to produce the actual rounding (similarly to the previous case, this makes the size of the roundings to automatically adapt to the local size and shape of the processed cell).

A *side* face (blue in Fig. 13) is produced only when all the four corresponding $\mathcal{M}_V$ vertices are exposed. A set of three *corner* faces (red in Fig. 13) is only produced when the corresponding cell vertex is exposed. A set of two *edge* faces (green in Fig. 13) is only produced if either or both the vertices on the corresponding cell edge are exposed; when only one is exposed, then the two faces are reduced to triangles, and the unexposed vertex is not displaced and kept to the original vertex position (Fig. 13, bottom). In this way, the tubular gaps around edges are artificially closed going toward the interior parts, leaving $\mathcal{M}_S$ geometrically watertight.

Similarly to the fissure mode, we willingly introduce an approximation to improve rendering performances. The visual impact of such an approximation is completely or almost completely negated both by occlusions and by lack of light reaching the affected parts.

A vertex is added to $\mathcal{M}_S$ only if it belongs to at least one produced face. The vertices of the side face (blue in Fig. 13) are assigned constant normals, making that face appear visually flat. The normal interpolation is limited to edges and corner faces (green and red in Fig. 13).

10.4. Supported file formats

HexaLab supports two among the most widely used file formats for the exchange of hex-meshes, namely the MEDIT format [73] (.mesh filename extension) and the VTK library format [74] (.vtk filename extension). Currently, HexaLab importer parses only the hexahedral elements expressed in these two formats, ignoring any other polyhedra (e.g., tetrahedra, pyramids and wedges).

10.5. Batch processing

HexaLab can load a zipped archive containing a collection of hex-meshes, and produces a zipped archive containing one screenshot (and one quality histogram) for each model, all sharing the same settings.

10.6. Employed Tools

The back-end, that deals with mesh input/output and geometric and topologic analysis, is developed in *C++* using *Eigen* library [75] for the linear algebra computations; for allowing the execution of C++ code on the browser client we used *Emscripten* [76] to "transpile" it in *asm.js* (a low-level subset of JavaScript) that is recognized and efficiently executed by modern js engines. The front-end, shown in Fig. 1, is developed directly in JavaScript, using standard web tools (HTML 5.0, CSS, AJAX, and *jQuery*) for the GUI, *webGL* and *Three.js* for the rendering and the trackball, and *Plotly.js* [77] for the graph plots. The code has no other dependencies. We use *GitHub* as an open-source repository for the code and the meshes and, thanks to the fact that it is a pure client web application, also for the web hosting.

11. Conclusions

We presented a novel tool for interactive hexahedral mesh visualization and first analysis, which combines a number of highly customizable tools to explore different features and characteristics of the inspected mesh. It produces readable images which convey both shape, quality, and the topological structure of the inspected hex-meshes, including its internal parts, plus simple numeric measures in form of graphs and data (according to a number of widely accepted measures). To this end, also HexaLab introduces several new visualization modalities specifically designed for this purpose, and employs a real-time global illumination model. It also provides direct access to a repository of results from several recent State-of-the-Art hex-mesh creation and processing solutions, thus easing further research by providing an easy way to compare against them. All data and images produced with HexaLab are easily reproducible and can be streamlined, specifically all the images shown in this paper can be re-created by loading into HexaLab the appropriate model and dragging the png file of each figure over the application window.

The tool is immediately available to researchers and practitioners in the form of an easily accessible 3D web-application (which is cross-platform, cross-browser, and cross-vendor, and requires no installation), and as an Open-Source project. We expect that it can be employed, in research, as a tool to gain insights on hex-meshes and therefore on the algorithms to produce, manipulate, and process them, and also to help the dissemination of engineering and scientific results (by producing high quality images fit for scientific articles and presentations).

11.1. Current limitations and future work

As mentioned in Sec. 4, one main planned activity consists in keeping the repository of State-of-The-Art hex-meshes up-to-date. There are several directions in which HexaLab can be expanded.

Generalization to polyhedral meshes: currently, HexaLab only supports pure hex-meshes. Other important classes of polyhedral meshes include (pure) tetrahedral meshes, and hybrid hex-dominant meshes. These classes are the focus of recent research works (e.g. [78] and [28] respectively). Many of the mechanisms employed by HexaLab are, in principle, extendible to them, including the filtering tools, the visualization modes, the internal data structures. Quality measures would have to be adapted, and it is not clear how to do so for hex-dominant meshes.

GUI improvements: in this project, we did not focus on GUI design, although preliminary testing with fellow researchers indicates that our GUI performs satisfactorily for our purposes. A deeper design analysis, including user studies, can be performed in the future to provide HexaLab with an improved interface. Note that this tool is intended for practitioners (modelers, architects, structural engineers, researchers on fields such as geometric processing and 3D computer graphics) and therefore its GUI targets expert users.

Improved visualization of the global structure: currently, Hexa-Lab highlights irregular edges, but additional mechanisms could be used to communicate the global structure of the inspected hex-mesh. For example, future versions could compute and use the base complex, or integrate the advanced visualization methods proposed in [79], which target the visualization of the (potentially intricate) whole structure of a hex-mesh.

Dynamic datasets: currently, HexaLab only supports static meshes. There is a research interest on dynamic datasets, which could be for example provided for visualization as different key-frame meshes sharing the same connectivity but different geometry, to be interpolated.

Attributes: HexaLab currently focuses on the shape of the mesh only, and its visualization disregards attributes which are often defined on its elements (such as vertices, or cells).

Trackball: HexaLab currently employs a standard sphere-based trackball to let the user orient the mesh, but a "generalized trackball" [80] could be employed in its place.

Acknowledgments

This work was partially funded by the EU ERC Advanced Grant CHANGE, agreement No 694515, and by Italian MIUR, project "DSURF" (PRIN 2015B8TRFM).

References

[1] Y. Li, Y. Liu, W. Xu, W. Wang, B. Guo, All-hex meshing using singularity-restricted field, ACM Trans. Graph. 31 (6) (2012) 177:1–177:11. doi:10.1145/2366145.2366196.

[2] G. Cherchi, M. Livesu, R. Scateni, Polycube Simplification for Coarse Layouts of Surfaces and Volumes, Computer Graphics Forumdoi:10.1111/cgf.12959.

[3] E. Wang, T. Nelson, R. Rauch, Back to elements-tetrahedra vs. hexahedra, in: Proceedings of the 2004 International ANSYS Conference, ANSYS Pennsylvania, 2004.

[4] S. J. Owen, A survey of unstructured mesh generation technology., in: IMR, 1998, pp. 239–267.

[5] T. Blacker, Meeting the challenge for automated conformal hexahedral meshing, in: 9th international meshing roundtable, 2000, pp. 11–20.

[6] R. Schneiders, A grid-based algorithm for the generation of hexahedral element meshes, Engineering with Computers 12 (3) (1996) 168–177. doi:10.1007/BF01198732.

[7] H. Lin, S. Jin, H. Liao, Q. Jian, Quality guaranteed all-hex mesh generation by a constrained volume iterative fitting algorithm, Comput. Aided Des. 67 (C) (2015) 107–117. doi:10.1016/j.cad.2015.05.004.

[8] L. Maréchal, Advances in Octree-Based All-Hexahedral Mesh Generation: Handling Sharp Features, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 65–84. doi:10.1007/978-3-642-04319-2_5.

[9] X. Gao, D. Panozzo, W. Wang, Z. Deng, G. Chen, Robust structure simplification for hex re-meshing, ACM Transactions on Graphics (TOG) 36 (6) (2017) 185.

[10] X. Gao, Z. Deng, G. Chen, Hexahedral mesh re-parameterization from aligned base-complex, ACM Transactions on Graphics (TOG) 34 (4) (2015) 142.

[11] M. Kremer, D. Bommes, I. Lim, L. Kobbelt, Advanced Automatic Hexahedral Mesh Generation from Surface Quad Meshes, Springer International Publishing, Cham, 2014, pp. 147–164. doi:10.1007/978-3-319-02335-9_9.

[12] X. Gao, T. Martin, S. Deng, E. Cohen, Z. Deng, G. Chen, Structured volume decomposition via generalized sweeping, IEEE transactions on visualization and computer graphics 22 (7) (2016) 1899–1911.

[13] M. Livesu, A. Sheffer, N. Vining, M. Tarini, Practical hex-mesh optimization via edge-cone rectification, ACM Transactions on Graphics (Proc. SIGGRAPH 2015) 34 (4). doi:10.1145/2766905.

[14] M. Livesu, M. Attene, G. Patané, M. Spagnuolo, Explicit cylindrical maps for general tubular shapes, Computer-Aided Design 90 (2017) 27 – 36. doi:10.1016/j.cad.2017.05.002.

[15] M. Livesu, A. Muntoni, E. Puppo, R. Scateni, Skeleton-driven adaptive hexahedral meshing of tubular shapes, in: Proceedings of the 24th Pacific Conference on Computer Graphics and Applications, PG '16, 2016, pp. 237–246.

[16] M. Tarini, K. Hormann, P. Cignoni, C. Montani, Polycubemaps, in: ACM transactions on graphics (TOG), Vol. 23, ACM, 2004, pp. 853–860.

[17] X.-M. Fu, C.-Y. Bai, Y. Liu, Efficient volumetric polycube-map construction, in: Computer Graphics Forum, Vol. 35, Wiley Online Library, 2016, pp. 97–106.

[18] J. Huang, T. Jiang, Z. Shi, Y. Tong, H. Bao, M. Desbrun, L1-based construction of polycube maps from complex shapes, ACM Transactions on Graphics (TOG) 33 (3) (2014) 25.

[19] M. Livesu, N. Vining, A. Sheffer, J. Gregson, R. Scateni, Polycut: Monotone graph-cuts for polycube base-complex construction, Transactions on Graphics (Proc. SIGGRAPH ASIA 2013) 32 (6). doi:10.1145/2508363.2508388.

[20] X. Fang, W. Xu, H. Bao, J. Huang, All-hex meshing using closed-form induced polycube, ACM Trans. Graph. 35 (4) (2016) 124:1–124:9. doi:10.1145/2897824.2925957.

[21] M. Nieser, U. Reitebuch, K. Polthier, Cubecover– parameterization of 3d volumes, Computer Graphics Forum 30 (5) (2011) 1397–1406. doi:10.1111/j.1467-8659.2011.02014.x.

[22] T. Jiang, J. Huang, Y. Wang, Y. Tong, H. Bao, Frame field singularity correctionfor automatic hexahedralization, IEEE Transactions on Visualization and Computer Graphics 20 (8) (2014) 1189–1199.

[23] N. Kowalski, F. Ledoux, P. Frey, Smoothness driven frame field generation for hexahedral meshing, Computer-Aided Design 72 (2016) 65–77.

[24] M. Lyon, D. Bommes, L. Kobbelt, Hexex: robust hexahedral mesh extraction, ACM Transactions on Graphics (TOG) 35 (4) (2016) 123.

[25] N. Ray, D. Sokolov, B. Lévy, Practical 3d frame field generation, ACM Transactions on Graphics (TOG) 35 (6) (2016) 233.

[26] J. Huang, Y. Tong, H. Wei, H. Bao, Boundary aligned smooth 3d cross-frame field, in: ACM transactions on graphics (TOG), Vol. 30, ACM, 2011, p. 143.

[27] H. Liu, P. Zhang, E. Chien, J. Solomon, D. Bommes, Singularity-constrained octahedral fields for hexahedral meshing, ACM Trans. Graph. 37 (4) (2018) 93:1–93:17. doi:10.1145/3197517.3201344.

[28] X. Gao, W. Jakob, M. Tarini, D. Panozzo, Robust hex-dominant mesh generation using field-guided polyhedral agglomeration, ACM Trans. Graph. 36 (4) (2017) 114:1–114:13. doi:10.1145/3072959.3073676.

[29] D. Sokolov, N. Ray, L. Untereiner, B. Lévy, Hexahedral-dominant meshing, ACM Trans. Graph. 35 (5) (2016) 157:1–157:23. doi:10.1145/2930662.

[30] C. Geuzaine, J.-F. Remacle, Gmsh: A 3-d finite element mesh generator with built-in pre-and post-processing facilities, International journal for numerical methods in engineering 79 (11) (2009) 1309–1331.

[31] U. Ayachit, The paraview guide: a parallel visualization application.

[32] Y. Zheng, R. W. Lewis, D. T. Gethin, Feview: An interactive visualization tool for finite elements, finite Elements in Analysis and Design 19 (4) (1995) 261–294.

[33] B. Levy, Geogram, http://alice.loria.fr/software/geogram/.

[34] R. A. Computer Graphics Group Aachen, Openvolumemesh: A generic and versatile index-based data structure for polytopal meshes, http://www.openvolumemesh.org.

[35] M. Livesu, cinolib: a generic programming header only c++ library for processing polygonal and polyhedral meshes, https://github.com/mlivesu/cinolib/ (2017).

[36] B. Levy, Graphite, http://alice.loria.fr/software/graphite/.

[37] M. L. Brewer, L. F. Diachin, P. M. Knupp, T. Leurent, D. J. Melander, The mesquite mesh quality improvement toolkit., in: IMR, 2003.

[38] R. M. Hanson, Jmol–a paradigm shift in crystallographic visualization, Journal of Applied Crystallography 43 (5) (2010) 1250–1260.

[39] M. Callieri, R. M. Andrei, M. Di Benedetto, M. Zoppè, R. Scopigno, Visualization methods for molecular studies on the web platform, in: Proceedings of the 15th International Conference on Web 3D Technology, ACM, 2010, pp. 117–126.

[40] A. S. Rose, P. W. Hildebrand, Ngl viewer: a web application for molecular visualization, Nucleic acids research 43 (W1) (2015) W576–W579.

[41] H. Li, K.-S. Leung, T. Nakane, M.-H. Wong, iview: an interactive webgl visualizer for protein-ligand complex, BMC bioinformatics 15 (1) (2014) 56.

[42] A. Garcıa-Alonso, Volume visualization tools for medical applications in ubiquitous platforms, eHealth 360: International Summit on eHealth, Budapest, Hungary, June 14-16, 2016, Revised Selected Papers 181 (2016) 443.

[43] J. Congote, A. Segura, L. Kabongo, A. Moreno, J. Posada, O. Ruiz, Interactive visualization of volumetric data with webgl in real-time, in: Proceedings of the 16th International Conference on 3D Web Technology, Web3D '11, ACM, New York, NY, USA, 2011, pp. 137–146. doi:10.1145/2010425.2010449.

[44] N. F. Polys, A. Wood, P. Shinpaugh, Cross-platform presentation of interactive volumetric imagery.

[45] M. Potenziani, M. Callieri, M. Dellepiane, M. Corsini, F. Ponchio, R. Scopigno, 3dhop: 3d heritage online presenter, Computer & Graphics 52 (2015) 129–141.
URL http://vcg.isti.cnr.it/Publications/2015/PCDCPS15

[46] R. Scopigno, M. Callieri, M. Dellepiane, F. Ponchio, M. Potenziani, Delivering and using 3d models on the web: are we ready?, Virtual Archaeology Review 8 (17) (2017) 1–9.
URL http://vcg.isti.cnr.it/Publications/2017/SCDPP17

[47] J. Gregson, A. Sheffer, E. Zhang, All-hex mesh generation via volumetric polycube deformation, Computer Graphics Forum 30 (5) (2011) 1407–1416. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2011.02015.x, doi:10.1111/j.1467-8659.2011.02015.x.

[48] M. Livesu, A. Sheffer, N. Vining, M. Tarini, Practical hex-mesh optimization via edge-cone rectification, ACM Trans. Graph. 34 (4) (2015) 141:1–141:11. doi:10.1145/2766905.

[49] X. M. Fu, C. Y. Bai, Y. Liu, Efficient volumetric polycube map construction, Computer Graphics Forum 35 (7) 97–106.

[50] X. Gao, T. Martin, S. Deng, E. Cohen, Z. Deng, G. Chen, Structured volume decomposition via generalized sweeping, IEEE Trans. Vis. Comput. Graph 22 (7) (2016) 1899–1911. doi:10.1109/TVCG.2015.2473835.

[51] F. Shang, Y. Gan, Y. Guo, Hexahedral mesh generation via constrained quadrilateralization, PLOS ONE 12 (5) (2017) 1–27. doi:10.1371/journal.pone.0177603.
URL https://doi.org/10.1371/journal.pone.0177603

[52] H. Wu, S. Gao, R. Wang, M. Ding, A global approach to multi-axis swept mesh generation, Procedia Engineering 203 (2017) 414 – 426, 26th International Meshing Roundtable, IMR26, 18-21 September 2017, Barcelona, Spain. doi:10.1016/j.proeng.2017.09.817.

[53] H. Wu, S. Gao, R. Wang, J. Chen, Fuzzy clustering based pseudo-swept volume decomposition for hexahedral meshing, Computer-Aided Design 96 (2018) 42 – 58. doi:10.1016/j.cad.2017.10.001.

[54] G. Cherchi, P. Alliez, R. Scateni, M. Lyon, D. Bommes, Selective padding for polycube-based hexahedral meshing, Computer Graphics ForumarXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13593, doi:10.1111/cgf.13593.
URL https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13593

[55] M. S. Langer, H. H. Bülthoff, Perception of shape from shading on a cloudy day, JOURNAL OF THE OPTICAL SOCIETY OF AMERICA A 11 (11) (1999) 467–478.

[56] M. Tarini, P. Cignoni, C. Montani, Ambient occlusion and edge cueing for enhancing real time molecular visualization, IEEE Transactions on Visualization and Computer Graphics 12 (5) (2006) 1237–1244. doi:10.1109/TVCG.2006.115.

[57] À. Méndez-Feliu, M. Sbert, From obscurances to ambient occlusion: A survey, The Visual Computer 25 (2) (2009) 181–196. doi:10.1007/s00371-008-0213-4.

[58] H. Landis, Production-ready global illumination, Siggraph course notes 16 (2002) (2002) 11.

[59] S. Zhukov, A. Iones, G. Kronin, An ambient light illumination model, in: G. Drettakis, N. Max (Eds.), Rendering Techniques '98, Springer Vienna, Vienna, 1998, pp. 45–55.

[60] L. Bavoil, M. Sainz, R. Dimitrov, Image-space horizon-based ambient occlusion, in: ACM SIGGRAPH 2008 Talks, SIGGRAPH '08, ACM, New York, NY, USA, 2008, pp. 22:1–22:1. doi:10.1145/1401032.1401061.

[61] M. Mittring, Finding next gen: Cryengine 2, in: ACM SIGGRAPH 2007 Courses, SIGGRAPH '07, ACM, New York, NY, USA, 2007, pp. 97–121. doi:10.1145/1281500.1281671.

[62] T. Ritschel, T. Grosch, H.-P. Seidel, Approximating dynamic global illumination in image space, in: Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games, I3D '09, ACM, New York, NY, USA, 2009, pp. 75–82. doi:10.1145/1507149.1507161.

[63] C. Stimpson, C. Ernst, P. Knupp, P. Pébay, D. Thompson, The verdict library reference manual, Sandia National Laboratories Technical Report 9.

[64] X. Gao, J. Huang, K. Xu, Z. Pan, Z. Deng, G. Chen, Evaluating hex-mesh quality metrics via correlation analysis, Computer Graphics Forum (SGP 2017).

[65] B. Li, X. Li, K. Wang, H. Qin, Generalized polycube trivariate splines, in: 2010 Shape Modeling International Conference, 2010, pp. 261–265. doi:10.1109/SMI.2010.40.

[66] C. Armstrong, H. J. Fogg, C. Tierney, T. Robinson, Common themes in multi-block structured quad/hex mesh generation 124 (2015) 70–82.

[67] A. Leff, J. T. Rayfield, Web-application development using the model/view/controller design pattern, in: Proceedings of the 5th IEEE International Conference on Enterprise Distributed Object Computing, EDOC '01, IEEE Computer Society, Washington, DC, USA, 2001, pp. 118–.
URL http://dl.acm.org/citation.cfm?id=645344.650161

[68] G. Damiand, Combinatorial maps, in: CGAL User and Reference Manual, 4.10 Edition, CGAL Editorial Board, 2017.
URL http://doc.cgal.org/4.10/Manual/packages.html#PkgCombinatorialMapsSummary

[69] P. Kraemer, L. Untereiner, T. Jund, S. Thery, D. Cazier, Cgogn: n-dimensional meshes with combinatorial maps, in: Proceedings of the 22nd International Meshing Roundtable, IMR 2013, October 13-16, 2013, Orlando, FL, USA, 2013, pp. 485–503. doi:10.1007/978-3-319-02335-9_27.

[70] P. Lienhardt, Topological models for boundary representation: A comparison with n-dimensional generalized maps, Comput. Aided Des. 23 (1) (1991) 59–82. doi:10.1016/0010-4485(91)90082-8.

[71] P. Lienhardt, N-dimensional generalized combinatorial maps and cellular quasi-manifolds., Int. J. Comput. Geometry Appl. 4 (3) (1994) 275–324. doi:10.1142/S0218195994000173.

[72] K. Hormann, M. Tarini, A quadrilateral rendering primitive, in: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, ACM, 2004, pp. 7–14.

[73] P. Frey, MEDIT : An interactive Mesh visualization Software, Tech. Rep. RT-0253, INRIA.

[74] W. J. Schroeder, B. Lorensen, K. Martin, The visualization toolkit: an object-oriented approach to 3D graphics, Kitware, 2004.

[75] G. Guennebaud, B. Jacob, et al., Eigen v3, http://eigen.tuxfamily.org (2010).

[76] A. Zakai, Emscripten: An llvm-to-javascript compiler, in: Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion, OOPSLA '11, ACM, New York, NY, USA, 2011, pp. 301–312. doi:10.1145/2048147.2048224.

[77] P. T. Inc., Collaborative data science (2015).
URL https://plot.ly

[78] Y. Hu, Q. Zhou, X. Gao, A. Jacobson, D. Zorin, D. Panozzo, Tetwild - tetrahedral meshing in the wild, ACM Transactions on Graphics (to appear).

[79] K. Xu, G. Chen, Hexahedral mesh structure visualization and evaluation, IEEE Transactions on Visualization and Computer Graphics 25 (1) (2019) 1173–1182. doi:10.1109/TVCG.2018.2864827.

[80] L. Malomo, P. Cignoni, R. Scopigno, Generalized trackball for surfing over surfaces, in: STAG: Smart Tools and Apps for Graphics, Eurographics, 2016.
URL http://vcg.isti.cnr.it/Publications/2016/MCS16