EUROGRAPHICS 2019 / P. Alliez and F. Pellacini
(Guest Editors)

Volume 38 (2019), Number 2

Dual Sheet Meshing: An Interactive Approach to Robust Hexahedralization

Kenshi Takayama

National Institute of Informatics

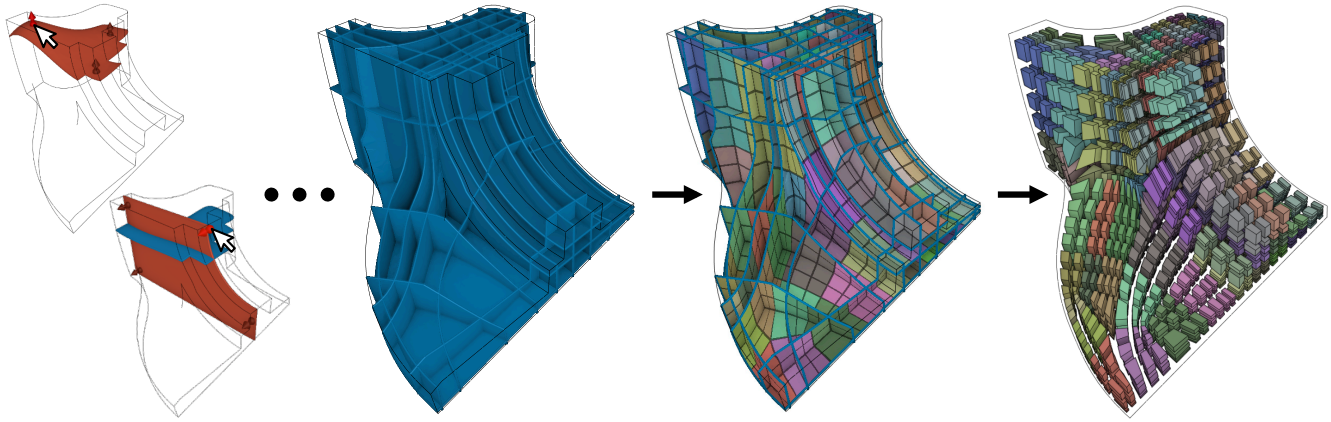


Figure 1: *In our approach, the user interactively designs a set of mutually intersecting surfaces called* dual sheets*, which is then primalized by the system to generate an all-hex mesh.*

Abstract

*The combinatorial dual of a hex mesh induces a collection of mutually intersecting surfaces (*dual sheets*). Inspired by Campen et al.'s work on quad meshing [CBK12, CK14], we propose to directly generate such dual sheets so that, as long as the volume is properly partitioned by the dual sheets, we are guaranteed to arrive at a valid all-hex mesh topology. Since automatically generating dual sheets seems much harder than the 2D counterpart, we chose to leave the task to the user; our system is equipped with a few simple 3D modeling tools for interactively designing dual sheets. Dual sheets are represented as implicit surfaces in our approach, greatly simplifying many of the computational steps such as finding intersections and analyzing topology. We also propose a simple algorithm for primalizing the dual graph where each dual cell, often enclosing singular edges, gets mapped onto a* reference polyhedron *via harmonic parameterization. Preservation of sharp features is simply achieved by modifying the boundary conditions. We demonstrate the feasibility of our approach through various modeling examples.*

CCS Concepts

• ***Computing methodologies*** → *Mesh models; Mesh geometry models; Volumetric models;*

1. Introduction

Methods for generating boundary-aligned high-quality hex meshes are of high demand for many fields of science and engineering where numerical analysis of 3D volumetric domains is involved. One of the major challenges here is to ensure that all the cells in the mesh are hexahedral (i.e., all-hex), a property frequently required in practice. Despite the active research in the recent years, fully automatic generation of high-quality all-hex meshes for general shapes is still difficult.

The basic motivation of our work comes from the fact that the combinatorial dual of a hex mesh induces a structure where many surfaces (*dual sheets*) intersect with each other, so by reversing this process, we should be able to obtain an all-hex mesh by somehow generating a set of mutually intersecting dual sheets and taking its combinatorial dual. This idea is directly inspired by Campen et al.'s work on quad meshing [CBK12, CK14]. Generalizing their approach to 3D seems difficult, however, because as opposed to the 2D case where dual loops, being 1-manifold, always have the same cyclic topology, dual sheets, being 2-manifold, can have arbitrarily

complex surface topologies. This enlarges the search space considerably, and it seems far from straightforward to design what is analogous to their anisotropic geodesic algorithm.

Our choice is to leave the automatic generation of an optimal set of dual sheets as an open problem, and instead ask the user to design the dual sheets through interactive 3D modeling. Our focus in this work is on the design of a simple and robust system that can generate all-hex meshes with some amount of manual labor (Figure 1), which we believe is of a certain value in some context. Our next choice is to represent dual sheets as zero isosurfaces of implicit functions, which not only eases the task of creating dual sheets of various surface topologies from the user's perspective, but also greatly simplifies many of the computational steps in the system such as finding intersections and analyzing topology. We also propose a simple algorithm for primalizing the dual graph which directly maps each dual cell, often enclosing singular primal edges, to a *reference polyhedron* via harmonic parameterization. Preservation of sharp features is trivially handled in this scheme by modifying the boundary conditions. We release our implementation publicly for the sake of reproducibility.

2. Related work

For a general background on hex meshing, please refer to the recent surveys [AFTR15, YZL15]. There are quite some publications in the engineering and the computational geometry communities that focus on the dual structure of hex meshes, such as the Whisker Weaving algorithm [TBM96], the dual cycle elimination algorithm [MH99, KBLK13] and a proof of existence for a hex mesh given a boundary quad mesh [Eri14]. These works are generally either more theoretical or of limited applicability in practice, however, and to our knowledge, no attempts have been reported so far on generating the actual geometry of dual sheets and then primalizing them to obtain all-hex meshes as we do in this work.

In the graphics community, one of the two popular approaches to all-hex meshing is based on volumetric PolyCube mapping [GSZ11, HJS*14, FXBH16]. While being able to offer some robustness guarantees, this approach inherently limits the class of hex mesh topologies being supported to those having singularities only on the boundary surface, consequently limiting the attainable quality. The other promising approach is based on the generation of a smooth octahedral field [HTWB11, RSL16, SVB17] which guides the construction of an integer-grid map [NRP11] followed by tracing integer isolines [LBK16]. This approach is considered to have the highest potential in achieving the best possible quality in all-hex meshing by allowing internal singularities, with a caveat that the process may fail if the field contains inappropriate singularity types, which is an active research topic [LLX*12, JHW*14, LZC*18]. In contrast to these automated methods, at the cost of requiring manual user input, our method can robustly generate all-hex meshes of arbitrary topologies including internal singularities (except for a few restrictions, see Section 9).

In the context of improving the quality of existing hex meshes, Gao et al. [GDC15, GPW*17] also focused on the dual structure and used sheet-wise operations to reduce the structural complexity of hex meshes.

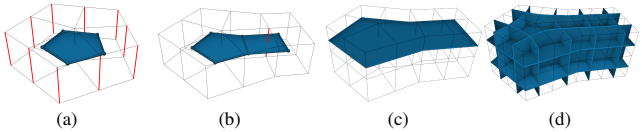


Figure 2: *The definition of a dual sheet. An edge of a primal hex mesh becomes a polygonal face in the dual graph (a), called a* dual face. *Each of the "parallel" edges surrounding that primal edge (highlighted in red) corresponds to a dual face adjacent to that dual face (b). The set of all dual faces associatively adjacent to each other forms a 2-manifold surface structure, called a* dual sheet *(c). A hex mesh can now be seen as a set of disjoint dual sheets intersecting with each other (d).*

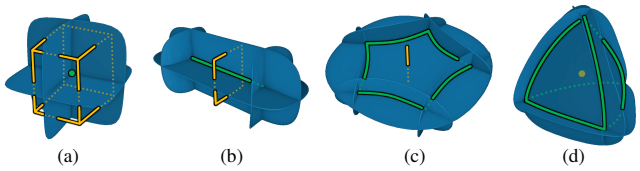
Our spirit in this work is also shared by some prior work on interactive quad remeshing [TPSHSH13, CK14, MTP*15, JTPSH15, ESCK16] where the user expresses through intuitive interfaces her particular application-specific intent that would otherwise not be captured by automatic algorithms. We believe similar tools for hex meshing to be valuable, but to our knowledge, no such attempt has been reported so far in the literature. There are many commercial packages (e.g., ANSYS [ANS18]) which have tools for decomposing volumes into blocks for hex meshing, but their technical details are largely unpublished.

3. Dual structure

Let us first introduce some terminology to make the subsequent explanation clear. Consider the combinatorial dual of a hex mesh. An edge of the hex mesh (a *primal edge*) becomes a polygonal face in the dual graph (a *dual face*) whose number of corners equals the number of primal hex cells incident to that primal edge (Figure 2a). We can then identify a set of other primal edges that are "parallel" to that primal edge (highlighted in red in Figure 2a), each corresponding to a dual face adjacent to the dual face of that primal edge (Figure 2b). By connecting all the dual faces that are adjacent to each other, we obtain a 2-manifold surface structure, called a *dual sheet* (Figure 2c). Just like a dual loop cannot have open ends inside the surface domain [CBK12], a dual sheet cannot have open boundaries inside the volumetric domain; i.e., it can either have boundary loops on the boundary surface of the volume, or form a closed surface inside the volume. The dual graph of a hex mesh can now be seen as a set of disjoint dual sheets intersecting with each other (Figure 2d).

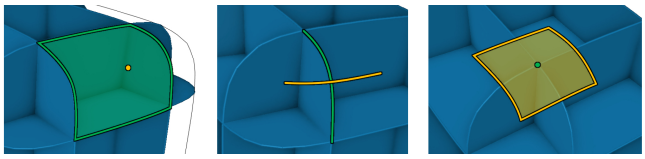
In this view, a primal hex cell corresponds to an intersection of three dual sheets (Figure 3a), called a *dual vertex*. A primal quad face corresponds to a segment of the intersection curve of two dual sheets bounded by two other sheets (Figure 3b), called a *dual edge*. A primal edge corresponds to a polygonal subregion of a dual sheet bounded by other intersecting dual sheets (Figure 3c), called a *dual face*. A primal vertex corresponds to a polyhedral subregion of the volume bounded by intersecting dual sheets (Figure 3d), called a *dual cell*.

Boundary elements. An element of the primal hex mesh at the boundary has an additional dual object associated with the boundary surface. A polygonal subregion of the boundary surface



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

Figure 3: *The relationships between elements of primal hex mesh (yellow) and elements of dual graph (green).*



(a) (b) (c)

Figure 4: *The definition of dual objects at the boundary surface.*

bounded by dual sheets, or equivalently, the intersection of a dual cell and the boundary surface, is called a *boundary dual face* (Figure 4a). Similarly, a segment of the boundary loop of a dual sheet bounded by two other dual sheets, or equivalently, the intersection of a dual face and the boundary surface, is called a *boundary dual edge* (Figure 4b). Finally, an intersection of boundary loops of two dual sheets, or equivalently, the intersection of a dual edge and the boundary surface, is called a *boundary dual vertex* (Figure 4c). A boundary dual face, a boundary dual edge, and a boundary dual vertex correspond to a boundary primal vertex, a boundary primal edge, and a boundary primal quad face, respectively. This fully agrees with the dual graph definition used in the context of quad meshing [CBK12].

Hex layout. As has been previously mentioned [GDC15], the singularity structure of a hex mesh induces another maximally simplified hex mesh topology, which we call *hex layout* (cf., the cited paper terms it *base complex*). Our focus is to generate the dual graph of this hex layout, thus minimizing the number of dual sheets considered. The final output hex mesh is obtained by regularly subdividing the hex layout cells (see Figure 5).

4. System description

The input to our system is a 3D triangle mesh describing the boundary surface of the volume to be hexahedralized. After some preprocessing (most notably, the tetrahedralization of the volume using TetGen [Si15] and the generation of a 4-RoSy field on the boundary surface using the instant meshes method [JTPSH15]), the user can start creating dual sheets. The user's workflow is divided into three steps: sheet editing, layout preview, and final adjustment (Figure 5). Please refer to the supplemental video for a demonstration of the system.

4.1. Sheet editing step

Our system offers three different tools for modeling dual sheets: *Freeform*, *Cylinder*, and *2D Sketch*.

Freeform. The user designs sheets as Hermite thin-plate spline implicit surfaces [MGV11] by placing Hermite (i.e., value and gradient) constraints shown as points with arrows. The user can create a new sheet with a single constraint by holding `Shift` and clicking on the boundary surface (Figure 6a). The new constraint is positioned at the clicked location, and its orientation is randomly set to one of the four directions of the 4-RoSy field. If the initial orientation is not what the user desired, she can rotate it about the surface normal by 90 degrees by pressing a key (Figure 6b). The user can adjust the position of the constraint by dragging the mouse, and change its orientation by dragging while holding `Ctrl` (Figure 6c).

The user can add more constraints to the currently selected sheet by holding `Shift` and clicking the boundary surface (Figure 6d). The new constraint is positioned at the clicked location, and its orientation is initialized with one among the four directions of the 4-RoSy field that is the closest to the current gradient of the sheet's implicit function.

By pressing `Space`, the user can hide the boundary surface and examine the shape of the dual sheets inside the volume (Figure 6e). The user can create constraints inside the volume by holding `Shift` and clicking the sheet surface (Figure 6f). The new constraint's orientation is initialized by the normal of the sheet surface. The adjustment of position and orientation of constraints inside the volume can be done using the same user interface as above (Figure 6g).

Cylinder & 2D Sketch. These tools are provided to support very common cases especially for man-made objects where some dual sheets can be viewed as an extrusion of a 2D profile curve (either a circle or other general shapes) along the normal direction of the 2D plane (Figure 7). With these tools, the user initially draws a freeform stroke on the screen. Upon completion, a plane parallel to the screen (called a *canvas*) is generated near the model, and all the drawn stroke points are projected onto the canvas. In the case of the *Cylinder* tool, the system calculates the center and radius of a circle that best fits the projected stroke points. The user can later adjust the radius of the cylinder by dragging the circle drawn on the canvas. In the case of the *2D Sketch* tool, the system downsamples the stroke points and generates a sparse set of Hermite constraints for 2D Hermite thin-plate spline implicit function. The user can manipulate these Hermite constraints using the same user interface for the *Freeform* tool. The user can also rotate and translate the entire canvas if necessary.

4.1.1. Required conditions for sheet configurations

When the user is finished with creating dual sheets, she presses a button and asks the system to compute the primal hex layout. During the process, the system throws an exception if any of the following situations is detected:

C1: Two sheets intersect at an acute angle (Figure 8a), or
C2: Four or more sheets intersect at almost the same location (Figure 8b). Although these tests are rather geometric instead of topological, and these problematic configurations do not prevent a valid hex layout topology from being generated, our system reports them as errors since they likely lead to undesirable results. The thresholds for these geometric tests can be adjusted depending on the application needs.

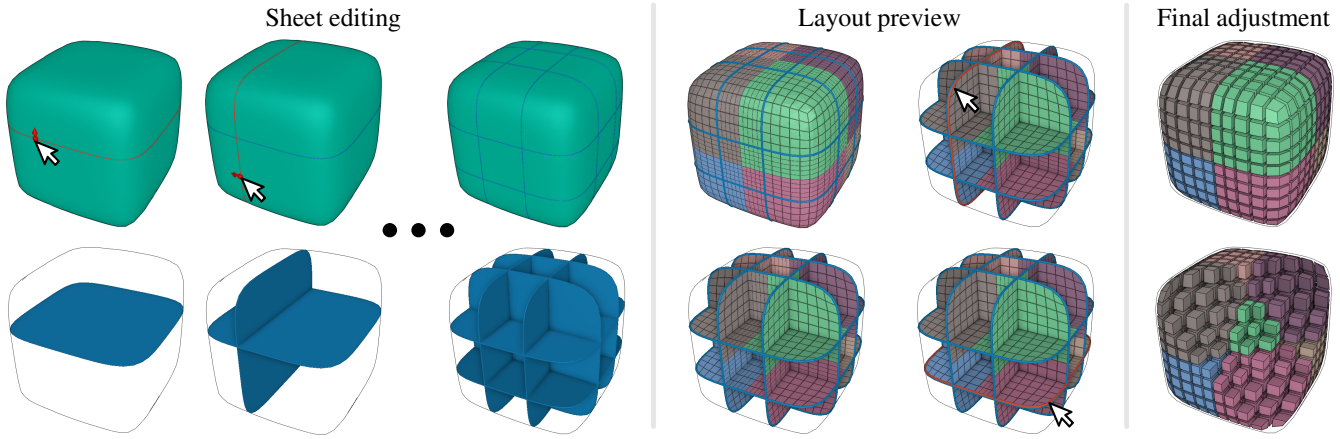


Figure 5: *An overview of our system.*

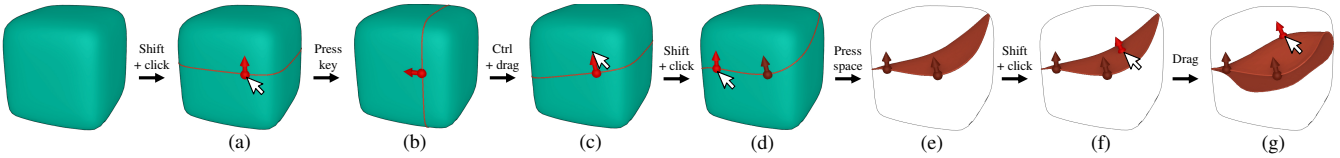


Figure 6: *Designing dual sheets using the* Freeform *tool.*

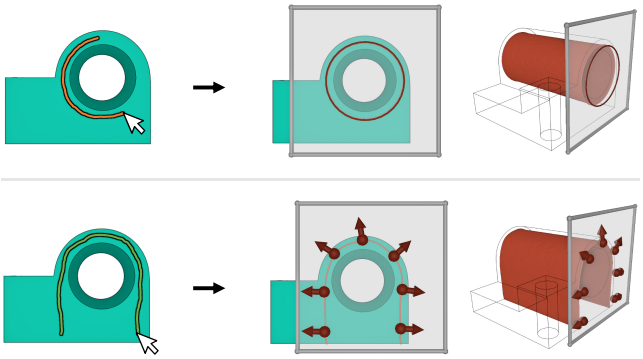


Figure 7: *Designing dual sheets using the* Cylinder *tool (top) and the* 2D Sketch *tool (bottom).*

C3: A volumetric subregion bounded by dual sheets does not have its corresponding primal layout vertex (Figure 8c). This situation can happen when a small portion of the volume is "cut off" by only one or two dual sheets. It means that the current sheet configuration is incomplete as a valid dual graph, which can be fixed by either removing those sheets from or adding more sheets to the problematic region.

C4: The same set of three dual sheets intersect at more than one locations with no other sheets separating them (Figure 8d). We treat this situation as anomaly because we assume the hex layout topology to be simple (i.e., each edge/face/cell should be identified by a unique pair/quadruple/octuple of vertices). This can be fixed by inserting a new sheet between those problematic intersections.

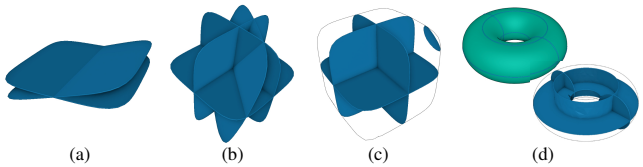


Figure 8: *Problematic sheet configurations treated as errors by our topology extraction algorithm.*

4.1.2. Additional requirements due to sharp features

If the boundary surface has sharp features to be preserved, the sheet configuration needs to satisfy an additional set of conditions. Before describing them, we first introduce the notion of a *feature graph*:

Feature graph. When the input triangle mesh is imported, its sharp edges are extracted based on the dihedral angles. A mesh vertex incident to one or three or more sharp edges is defined as a *feature node*, while a sequence of sharp edges running between two feature nodes or forming a closed loop without touching any feature nodes is defined as a *feature arc* (Figure 9). Each feature node represents a boundary vertex to be present in the final hex mesh, while each feature arc represents a sequence of boundary edges to be present in the final hex mesh. The feature graph can be further edited if desired; any manually selected edges can be made into feature arcs. Any vertex on a feature arc can be made into a feature node, splitting the arc into two. Conversely, a feature node incident to exactly two feature arcs can be eliminated, merging the two arcs into one. The user can also create completely isolated feature nodes, not incident to any feature arcs. Through manual editing of the feature

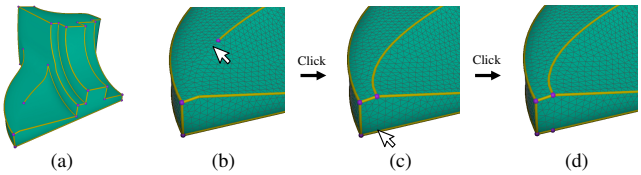


Figure 9: *A feature graph consisting of nodes (purple) and arcs (yellow), describing the edges and vertices to be preserved in the final hex mesh. After initialization based on the dihedral angles (a), the user can edit the graph freely as needed (b,c,d).*

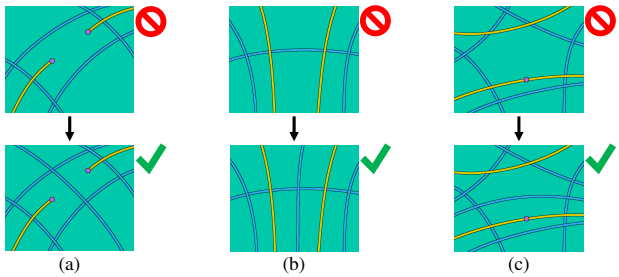


Figure 10: *Inappropriate configurations of dual sheets with respect to the feature graph (top), and their resolutions by inserting new sheets (bottom).*

graph, the user can enforce hard constraints on the boundary vertices and edges of the final hex mesh.

This definition of a feature graph naturally implies additional requirements for the dual sheet configuration as follows:

D1: A boundary dual face can enclose at most one feature node (Figure 10a), otherwise two or more feature nodes would get mapped to the same boundary layout vertex, breaking our assumption.

D2: A boundary dual edge can be intersected by at most one feature arc (Figure 10b), for the analogous reason as above.

The next condition is involved with feature arcs that *run through* boundary dual faces; we say that a feature arc *runs through* a boundary dual face if two of its constituent boundary dual edges are intersected by the same arc.

D3: If a feature arc runs through a boundary dual face, no other feature nodes nor feature arcs can be present inside the boundary dual face (Figure 10c). A feature arc running through a boundary dual face means that the corresponding boundary layout vertex must be lying somewhere on the arc. Having other feature nodes or feature arcs inside the same boundary dual face contradicts this assumption.

And finally, another simple requirement is:

D4: A feature arc must be intersected by at least one dual sheet, otherwise the feature arc would not be represented by any of the boundary edges of the final hex mesh.

The violation of any of the above conditions **D1**–**D4** can always be resolved by inserting a new sheet such that the relevant feature graph elements get separated. Note, however, that inserting new sheets may create new problematic configurations **C1**–**C4**.

When any of these errors is detected, the system reports the 3D location at which the error occurred, the type of the error, and the list of items involved in the error (IDs of sheets, feature arcs and nodes), in a basic textual form in the terminal window.

4.2. Layout preview step

Once all the required conditions are met, the system can complete the topology extraction step (Section 5), and proceeds to the remaining computational steps including the refinement of the tet mesh (Section 7.1) and the 2D parameterization of the dual faces (Section 6.1), which typically requires a bit of waiting.

In the next step of layout previewing, the edges of the final hex mesh are visualized as textures, while the partitioning of the volume into distinct layout cells is visualized using colors. The user's task in this step is to specify the number of edge subdivisions defined for each sheet, which is initialized according to a given target edge length. All the layout edges intersected by the same sheet must have the same number of subdivisions, and the resulting hex mesh is always conforming regardless of user input. The number of edge subdivisions can be specified for the front and back sides of each sheet separately; the specified number of edge subdivisions for the respective sides will determine the regular 3D lattices generated inside the corresponding dual cells (Section 6.3). When the user selects a sheet, the system renders the sheet's boundary loop twice with small offsets along the normal direction, and highlights the currently selected side in red (Figure 5).

As the user changes the number of edge subdivisions, the visualization is updated immediately since only a fragment shader program needs to be reexecuted with different parameters. If necessary, the user can go back to the previous step and modify the sheet configuration.

4.3. Final adjustment step

When the user is satisfied with the number of edge subdivisions for each dual sheet, she presses a button to let the system compute the 3D parameterization of the dual cells and generate the final hex mesh (Section 6.3). In the final adjustment step, the user can perform some trivial postprocessing operations such as global/local smoothing and manual repositioning of vertices. If desired, the user can go back to the previous step and readjust the edge subdivisions. Note that the final hex mesh generation is quicker this time, because the 3D parameterization of the dual cells was already computed.

5. Topology extraction

We present a simple and robust algorithm for analyzing the topological structure of the dual graph and finding intersections among dual sheets that fully make use of the implicit surface representation for the dual sheets.

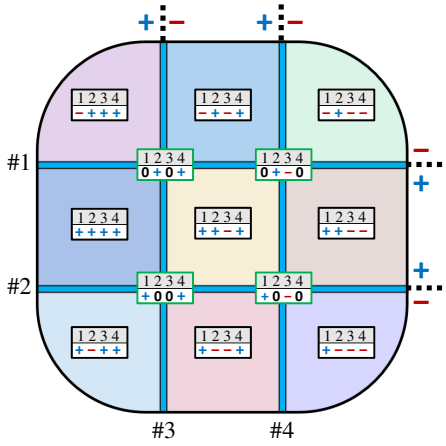


Figure 11: *A 2D example of element codes where a domain is divided by four sheets (#1–#4). Subregions bounded by the sheets are assigned their unique vertex codes (in boxes with black border), while intersections of two sheets are assigned their cell code (in boxes with green border).*

5.1. Element code

Our insight is that any point inside the volume (not exactly on any of the dual sheets) can be easily classified into its corresponding dual cell by looking at the signs of all the dual sheets' implicit functions. We define such a sequence of signs as a *vertex code*. Now, let us consider a pair of adjacent dual cells sharing a common dual face that is part of the i-th dual sheet. The vertex codes of these two dual cells differ only at one position corresponding to the i-th sheet, one having positive and the other having negative. Therefore, we can naturally define an *edge code* for this dual face by setting the value corresponding to the i-th sheet to zero. This treatment is consistent with the relation between the primal edge and the dual face, and the fact that the sheet function corresponding to the dual face will evaluate to zero at all points on the sheet. A *face code* and a *cell code* can be defined analogously; i.e., a face code and a cell code have two and three positions set to zero corresponding to the set of intersecting sheets that induce them, respectively. See Figure 11 for a 2D example.

5.2. Topology extraction algorithm

Using this notion of element codes, the topology extraction process becomes extremely simple. First, we piecewise-linearly discretize the dual sheets' implicit functions over the tet mesh, then compute all the dual sheets' intersections with all of the tets in the tet mesh. Next, for each tet, if it has intersections with three or more sheets, we examine all the triplets of the intersecting sheets and see if the intersection of the three planes corresponding to the triplet lies within this tet or not, which is trivial to check: suppose the three sheets' function values evaluated at the four vertices of the tet are f_i, g_i, and h_i for $1 \leq i \leq 4$. The intersecting point represented by the tet's barycentric coordinate λ_i satisfies:

$$\sum \lambda_i f_i = 0, \qquad \sum \lambda_i g_i = 0, \qquad \sum \lambda_i h_i = 0.$$

Together with the condition $\sum \lambda_i = 1$, the intersecting point can be easily computed by solving a mere 4×4 system of linear equations. If all the λ_i are positive, the intersecting point is found inside the tet. We then derive the code for the new hex cell by evaluating the function values of all but the three intersecting sheets (which are already known to be zero) at the location of the intersection. We can then derive the vertex code for each of the hex cell's eight vertices by substituting the three zeros in the cell code with the corresponding combination of positives and negatives. We use OpenVolumeMesh [KBK13] for handling the volumetric mesh topology; each time a new vertex code is encountered, a new vertex instance is created with `add_vertex`, and the mapping from the vertex code to the new vertex's handle is stored. After all the eight vertices' handles are identified, a new hex cell instance is created with `add_cell`. Note that the orientation of the hex cell needs to be made consistent throughout the entire mesh; we ensure this by checking the relative orientation of the three sheet functions' gradients at the intersection.

One strength of our algorithm is that it reliably detects all the intersecting points of three sheets (and analogously the intersecting curves of two sheets) regardless of how many distinct sheets intersect within the same tet. Note, however, that the tet mesh needs to be dense enough to be able to capture the geometric complexity of all of the sheets (e.g., a sheet making a sharp U-turn with some of its part coming close to another part of itself).

6. Primalization

For the 2D counterpart problem of quad layout primalization, Campen et al. [CBK12, CK14] used either the singularity of the orientation field if only one is enclosed in a dual face, or otherwise the geodesic center of the dual face, as the endpoints of anisotropic geodesics representing separatrices between the layout's singular vertices, and used the intersections of the separatrices as the positions of the layout's regular vertices. In our case, however, we could not simply generalize this approach to 3D because, as mentioned in Section 1, it seems far from straightforward to generalize their anisotropic geodesic algorithm which is designed to find 1-manifold objects, to our volumetric case where 2-manifold objects (i.e., the surface patches representing the hex layout faces) are sought for. Instead of exploring this direction, we developed a very different approach based on a simple idea of mapping a dual cell directly to a *reference polyhedron* via harmonic parameterization.

In the conventional setting for a harmonic parameterization in 2D, one maps a rectangular target domain on a 3D triangle mesh to a unit square in the 2D parameter space. By mapping the isolines of the UV coordinates back to the original 3D space, one obtains a regular 2D lattice on the target domain that smoothly follows the boundary. The same idea can be employed for the volumetric case to obtain a regular 3D lattice inside a rectangular volumetric domain. Our problem setting is similar to these, but in our case, our target domains have topology of general polygons in 2D and general polyhedra with degree-3 vertices in 3D. Our primalization algorithm consists of two steps: the 2D parameterization of dual faces which is performed when the user proceeds to the layout preview step (Section 4.2), and the 3D parameterization of dual cells

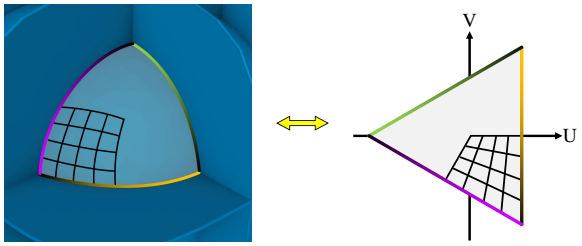


Figure 12: *2D parameterization of a triangular dual face onto the regular triangle. The correspondences along the boundary established by arc-length parameterization is visualized using color gradients. For illustrative purposes, a 4×4 lattice grid corresponding to the regular subregion of one corner (adjacent to the purple and yellow sides) is also shown.*

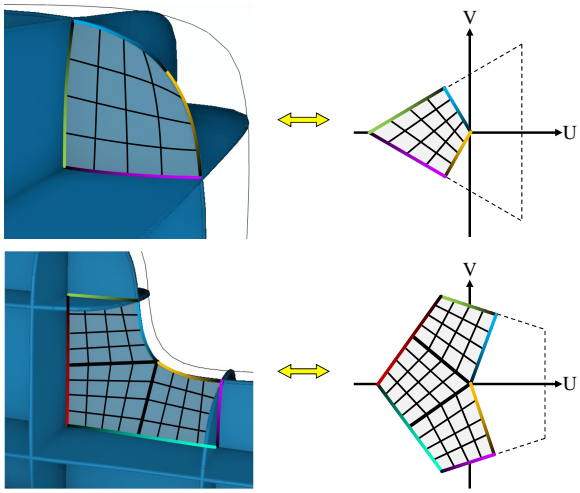


Figure 13: *2D parameterization of a dual face adjacent to the boundary surface for cases where $N = 3$ (top) and $N = 5$ (bottom).*

which is performed after the user confirms the generated hex layout (Section 4.3).

6.1. 2D parameterization of dual faces

Since a dual face (corresponding to a primal edge) is topologically equivalent to an N-sided polygon where N is the number of primal cells incident to the primal edge (we assume the edge to be inside the volume for now), we map it to a regular N-sided polygon in the 2D parameter space centered at the origin (Figure 12). We define the map as piecewise bilinear, where each of the N corners has its own bilinear map specified by the following four points:

- the corner itself,
- the midpoints of the two incident edges, and
- the centroid of the polygon (i.e., the origin).

We call such a rectangular region for bilinear interpolation defined for each corner as a *regular subregion*. Note that the harmonic parameterization itself is carried out in the standard fashion; each point on the dual edges surrounding the dual face gets assigned the corresponding UV coordinate along the boundary of the N-sided polygon in the parameter space via arc-length parameterization, and after solving the system of linear equations, each point on the dual face receives the resulting UV coordinate. If one wishes to generate a quad mesh over the dual face (which is not needed in our actual scenario), she would generate a 2D lattice for each regular subregion in the 2D parameter space, and map the lattice points back to the original 3D space.

For dual faces incident to the boundary: We map the dual face to a regular N-sided polygon in the 2D parameter space where N is the number of primal cells incident to the primal edge *plus two*. Intuitively, we can regard the dual face being mapped to the 2D parameter space as "missing" two consecutive regular subregions corresponding to the side of the boundary surface (Figure 13). Therefore, we can handle this case by modifying the specification of the boundary conditions in the above scheme; we map the boundary dual edge belonging to this dual face to a polyline consisting of the following three points:

- the midpoints of two edges adjacent to the boundary side, and
- the polygon's centroid (i.e., the origin).

Another modification is that we map the dual edges incident to the boundary dual edge to the halves of the corresponding line segments of the regular polygon, opposite to the boundary.

Finally, we also compute the 2D parameterization for each of the boundary dual faces (corresponding to the boundary primal vertices) in the exact same way. The 2D parameterization result will be used to specify the boundary conditions for the 3D parameterization, as explained below.

6.2. Reference polyhedron

In order to generalize the above idea to 3D, we need to come up with a "regular" polyhedron that is topologically equivalent to each dual cell, which we call a *reference polyhedron*. Each face of such a polyhedron is an N-sided polygon corresponding to a dual face belonging to the dual cell, and each vertex of the polyhedron has degree three (i.e., has three incident edges and faces) because it corresponds to a dual vertex which is an intersection of three sheets. Thanks to Liu et al.'s observation [LZC*18], we know there exist only eleven types of such polyhedra if we limit the degree of dual faces (or equivalently, the valence of primal edges) between three and five, which is a moderate restriction and is adopted here as well. We generated the actual geometry (i.e., the 3D vertex positions) of these polyhedra by making the edges as equilateral as possible (Figure 14); we initialized the vertex positions with some reasonable values, and performed interleaved iterations of averaging vertex positions and normalizing edge vectors until convergence (except for the one with signature (0,3,6) where we first performed the above iterations and then manually repositioned a pair of vertices outward so that the strong concavity around them is eliminated). Note that we center each reference polyhedron at the origin.

6.3. 3D parameterization of dual cells

We first explain our algorithm for the dual cells inside the volume. For each dual cell (corresponding to a primal vertex), we obtain its

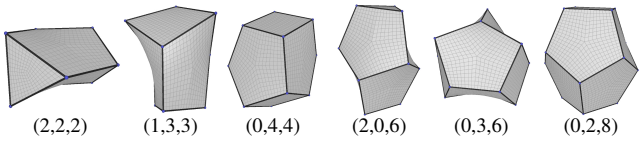


Figure 14: *The actual shapes of the reference polyhedra. The five trivial ones, regular polyhedra and prisms, are omitted. Since the faces are generally non-planar, their shapes are visualized by bilinearly interpolating vertex positions, edge midpoints, and face centroids.*

signature [LZC*18] by counting the valences of the incident primal edges, and use it to choose the corresponding reference polyhedron from the eleven types. Next, we find a mapping from the dual cell's vertices to the reference polyhedron's vertices using a graph isomorphism algorithm based on backtracking. Then, for each point on each dual face, we assign a UVW coordinate as the boundary condition of the 3D harmonic parameterization as follows (Figure 15): from the UV coordinate of the point on the dual face which was computed in the previous step (Section 6.1), we can determine which regular subregion of the N-sided polygon the point belongs to, by looking at its angular component in the polar coordinate representation. Furthermore, we can compute the bilinear interpolation parameter that reproduces the same UV coordinate by solving the inverse bilinear interpolation problem. Finally, we can obtain the UVW coordinate for this point by performing a bilinear interpolation with the obtained interpolation parameter against the following four points in the 3D parameter space:

- the reference polyhedron's corner corresponding to the regular subregion,
- the midpoints of the two edges of the reference polyhedron's face corresponding to the dual face and incident to the above-mentioned corner, and
- the centroid of the reference polyhedron's face corresponding to the dual face.

For dual cells at the boundary: Similar to the 2D case, we can regard the dual cell being mapped to the 3D parameter space as "missing" an entire layer of regular subregions corresponding to the side of the boundary surface. Therefore, we modify the boundary condition such that the boundary dual face is mapped to the cross-section of the reference polyhedron exposed after cutting off the layer of regular subregions, see Figure 16.

After the boundary conditions have been fully specified, we solve the system of linear equations and obtain a UVW coordinate for each point inside the dual cell. We finally obtain the output hex mesh by generating a 3D lattice over each regular subregion of each dual cell in the following way: for each lattice point, its corresponding UVW coordinate is obtained by performing a trilinear interpolation against the following eight points in the 3D parameter space (Figure 17):

- the reference polyhedron's corner corresponding to the regular subregion,
- the midpoints of the reference polyhedron's three edges incident to the above-mentioned corner,

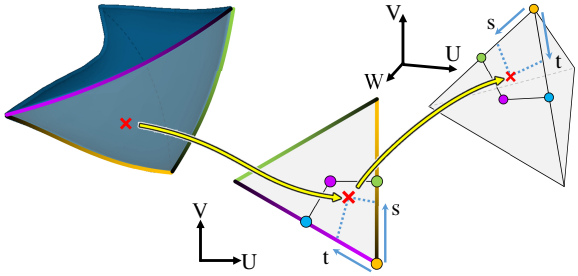


Figure 15: *3D parameterization of a dual cell with signature* $(4,0,0)$ *(left) onto the corresponding reference polyhedron, i.e., the regular tetrahedron (right). Each point on each of the constituent dual faces (red cross) has been assigned its UV coordinate after the 2D parameterization step (middle). By looking at the angle between the coordinate vector and the U axis, we can trivially know which regular subregion this point belongs to (four colored points). We also obtain the corresponding bilinear interpolation coefficients* (s,t) *by solving the inverse bilinear interpolation problem. The UVW coordinate assigned to this point as the boundary condition of 3D parameterization is then obtained by bilinearly interpolating the corresponding four points on the reference polyhedron.*

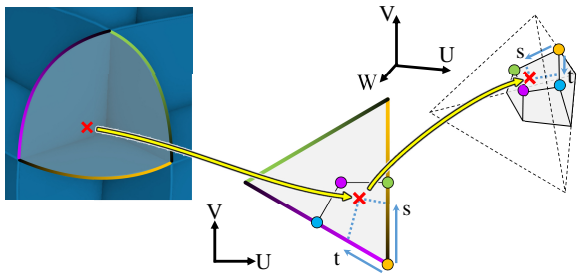


Figure 16: *Specifying the boundary conditions of 3D parameterization for a dual cell at the boundary surface.*

- the centroids of the reference polyhedron's three faces incident to the above-mentioned corner, and
- the centroid of the reference polyhedron (i.e., the origin).

Lastly, we map the interpolated UVW coordinate back to the original 3D space to obtain the XYZ coordinate for this lattice point.

Note that the found isomorphism between the dual cell vertices and the reference polyhedron's vertices is always ambiguous up to reflectional symmetry. In order to make the hex cells consistently oriented throughout the entire mesh, we check if the isomorphism flipped the vertex ordering or not, and use the proper vertex ordering when adding hex cells to the mesh.

6.4. Preservation of sharp features

Our scheme can easily support the preservation of sharp features by making slight modifications to the 2D parameterization step; no changes are needed for the 3D parameterization step.

For dual faces incident to the boundary: Due to the required condition **D2** (Section 4.1.2), a boundary dual edge can be intersected by at most one feature arc. We map such an intersected point to the

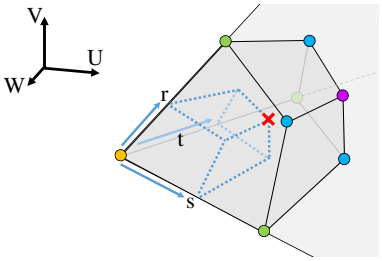


Figure 17: *Obtaining a UVW coordinate for a lattice point by trilinearly interpolating UVW coordinates of the reference polyhedron's corner (yellow), the edge midpoints (green), the face centroids (blue), and the centroid (purple).*

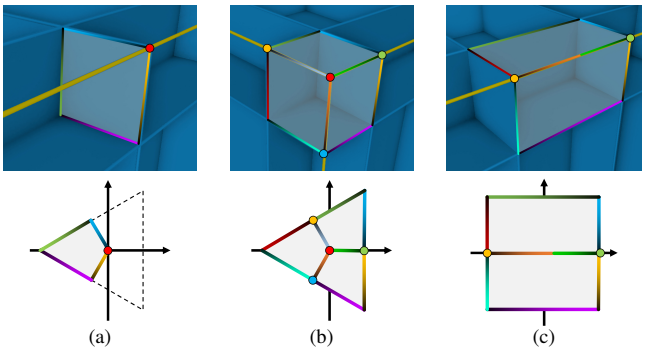


Figure 18: *Modifications to the 2D parameterization step to achieve feature preservation. Note that in (c), since the exact length-wise midpoint of the feature arc will generally be on an edge of the triangle mesh, no vertex is mapped to the origin. When splitting the feature arc into two, its constituent triangle mesh edges up to the above-mentioned edge are included in one part (orange) while the rest is included in the other (green).*

origin of the 2D parameter space, and map the two halves of the boundary dual edge to the corresponding segment using arc-length parameterization separately (Figure 18a).

For boundary dual faces: If a boundary dual face encloses a feature node (which is ensured to be unique due to **D1**), the corresponding point is mapped to the origin of the 2D parameter space. Each feature arc connected to the feature node is mapped to a segment in the 2D parameter space between the origin and the midpoint of the corresponding side of the N-sided polygon using arc-length parameterization (Figure 18b).

Otherwise, due to **D3**, there is one feature arc running through the boundary dual face. In this case, we split the feature arc roughly at its middle, and map the two halves to the corresponding segments in the 2D parameter space using arc-length parameterization (Figure 18c).

7. Implementation details

7.1. Generating a PLC for refining the tet mesh

The parameterization algorithm described in the previous section demands a refined tet mesh that fully conforms to all the intersections between the original tet mesh and the sheets and the intersections among the sheets (Figure 19). We again use TetGen [Si15] for this mesh refinement step which takes a piecewise linear complex (PLC) as input describing how the volume should be partitioned. In our case, each cross-section of a tet by a sheet (either a triangle or a quadrilateral) is potentially cut by arbitrarily many other sheets, resulting in a partitioning of the cross-section into many small facets, each corresponding to a distinct dual face. By making use of our function-based representation for the dual sheets, we devised a simple recursive algorithm to compute such a partitioning.

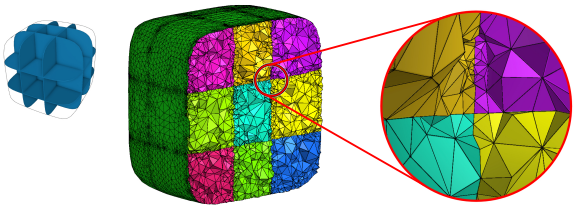


Figure 19: *A fully conforming refined tet mesh generated using TetGen.*

We observe that every point in the PLC must originate from one of the following:

- an intersection of an edge of the tet mesh and a sheet,
- an intersection between two sheets on a face of the tet mesh, or
- an intersection among three sheets within a cell of the tet mesh.

We first compute all these intersection points and store them in the PLC's point list, and while doing so, we store a mapping from a combination of a tet mesh element and a set of relevant sheets (e.g., "tet mesh face #10 and sheets #2 & #5") to the corresponding index into the point list.

Now, we introduce a data structure, denoted as P, representing a polygonal subregion of a tet's cross-section, that is to be further split by other sheets. P can hold arbitrary number of corners, and for each corner, it stores the function values of all the sheets as well as its corresponding index into the point list. For each segment between two corners, P also stores from which source the segment originates: either from a tet mesh face or from another sheet. Using this information, the function SPLITPOLYGON(P, j) computes where P gets split by the j-th sheet by looking at the function values at the corners, and also obtains the indices into the point list for the two intersections by using the information stored in each segment. It outputs two new polygons P_+ and P_- which are fed to the recursive function PROCESSPOLYGON(P, i, $j+1$). For each tet intersected by the i-th sheet, the recursive splitting process is started as PROCESSPOLYGON(P_{init}, i, 1) with the initial polygon P_{init} corresponding to the original cross-section (either a triangle or a quadrilateral). See Algorithm 1 for a pseudocode and Figure 20 for an example.

In addition to the facets on the dual sheets, we also need to handle the facets on the boundary surface, which is done in a similar fashion. Also note that we use TetGen's *region attributes* field to tell it which volumetric subregion defined by the PLC corresponds to which dual cell.

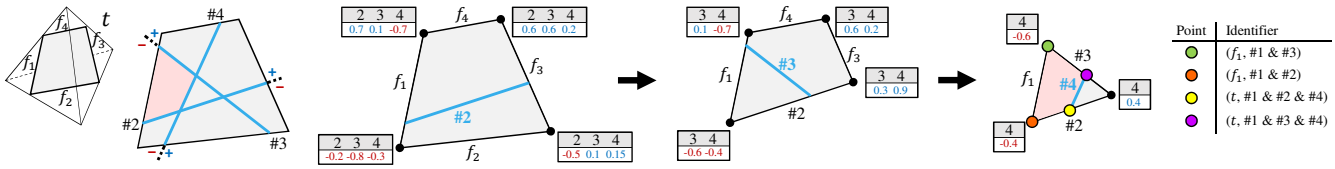


Figure 20: *Generating facets of a PLC by recursively splitting the cross-section of a tetrahedron by a sheet. Suppose a tet mesh cell t is intersected by sheet #1, producing a quadrilateral cross-section whose segments come from tet mesh faces $f_1, \cdots, f_4$. The cross-section is further split by three other sheets (#2–#4), and the process of one facet highlighted in red being generated is shown. Each time the polygon gets split by the next sheet, the function values of the other remaining sheets at the intersections are obtained by linear interpolation. Each segment stores from which source it originates (either a tet mesh face f_i or a sheet #i) which allows us to uniquely identify the intersected points in the PLC's point list.*

Algorithm 1 PROCESSPOLYGON(P, i, j)

```
1: if j > N then                          ▷ N is the number of sheets
2:     ADDPOLYGON(P);                     ▷ adds P to the list of facets
3: else if i = j or the j-th sheet has no intersection with P then
4:     PROCESSPOLYGON(P, i, j+1);
5: else
6:     P+, P− ← SPLITPOLYGON(P, j);
7:     PROCESSPOLYGON(P+, i, j+1);
8:     PROCESSPOLYGON(P−, i, j+1);
```

7.2. Merging hex mesh vertices shared by multiple subregions

When generating the 3D lattice for each regular subregion of each dual cell, we need to take into account that vertices at the boundary of the lattice are shared by the lattices of the neighboring regular subregions (either belonging to the same dual cell or to an adjacent dual cell). We implemented a simple scheme to obtain a unique handle for each of these shared vertices without having to explicitly check the relative orientation of the neighboring dual cells. Since every lattice point is generated by a trilinear interpolation of the UVW coordinates assigned to the eight corners of the lattice which correspond to the following eight primal layout elements:

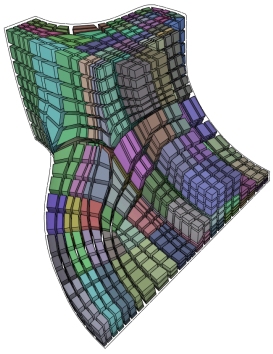
- a layout vertex,
- three layout edges,
- three layout faces, and
- a layout cell,

it can be uniquely identified by the set of the relevant primal layout elements along with their nonzero weights. For example, a lattice point topologically at the middle between a layout vertex v_i and the midpoint of a layout edge e_j is uniquely identified by a pair of *weighted handles*, $((v_i, 1/2), (e_j, 1/2))$. The weights are represented by rational numbers to circumvent the floating point rounding error that could otherwise occur in the equality test.

8. Results

We implemented our system using C++ and OpenGL. We use libigl [JP*18] for some basic geometry processing tasks such as harmonic parameterization. For the examples shown in this paper, the tetrahedralization of the input triangle mesh contains roughly 10k vertices, while the refined tet mesh contains roughly 150k to 200k vertices. On a laptop with a 2.9 GHz Intel Core i7 CPU, it takes roughly one to two minutes to generate the refined tet mesh and compute the 2D parameterization of dual faces, while it takes roughly one minute to compute the 3D parameterization of dual cells and generate the final hex mesh. The system consumes relatively large amount of RAM ($\sim$ 2 GB) mainly due to naively storing the values of dual sheets' implicit functions at each tet mesh vertex. We expect this issue to be mitigated by storing the function values only at vertices of tets intersected by dual sheets. We also expect the processing speed to be improved through parallelization.

All modeling examples presented here were created by the author. As a simple sanity check to confirm that our primalization algorithm works fine for all of the eleven signature types, we took each of the surface meshes shown in Figure 14 as input to our system, and successfully hexahedralized it (Figure 21).



Next, we tested our system against the FANDISK model which is notoriously difficult to hexahedralize because of its subtle features. Since creating a good hexahedral topology for this model from scratch seemed hard, we instead chose to reproduce the same topology (up to regular subdivision) as the one shown in Liu et al.'s paper [LZC*18] (inset). Starting from this reproduced topology, we modified some portion of it such that the manually selected subtle feature shown in Figure 9 is preserved (Figure 1). This usage pattern suggests a potential utility of our system in quickly reproducing a given hexahedral topology and experimenting with its variations.

Figure 22 shows some other modeling examples which were created without referring to any existing hexahedralizations. The BUNNY model was especially difficult to deal with for our current modeling interfaces; our system could benefit from the wealth of prior art on 3D modeling.

9. Limitations and future work

Our work is only a first step in the direction of dual-based all-hex meshing with some practical issues and rooms for improvement:

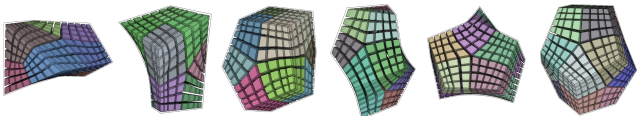


Figure 21: *A sanity check to confirm that our primalization algorithm works fine for all the signature types.*

- Generating hex meshes with mostly uniform edge lengths may be difficult using our system since it requires careful design of layout topology, a complex task entirely delegated to the user.
- The use of implicit representation for dual sheets, while beneficial in terms of simplicity, imposes some strong restrictions on the surface topology allowed for dual sheets. The strongest among them is that self-intersecting dual sheets cannot be created using our system, which are often relevant as evident in the NAUTILUS model (Figure 9a) identified by Viertel et al. [VSL16]. Our system also does not permit dual sheets that do not divide the volume into two or more regions (Figure 9b). One way to fix these issues would be to switch to using the explicit representation for dual sheets, where the implementation complexity as well as the runtime cost are expected to increase. Another way would be to develop some modifications to the implicit representation such that these issues could be avoided.
- Unlike in 2D, a mapping in 3D obtained via harmonic parameterization is not guaranteed to be bijective. We never encountered such a mapping that lacks bijectivity while testing our system, most likely because the boundary geometry of dual cells tended to be smooth and simple. Investigating when the mapping loses bijectivity and how to fix it, is left for future work.
- The required conditions described in Sections 4.1.1 and 4.1.2 may not be sufficient for inducing a valid hex topology and enforcing hard constraints expressed by a feature graph, respectively. We empirically identified these conditions by running into errors due to them not being satisfied. Determining if these conditions are sufficient or not is left for future work.
- The user interface for modeling dual sheets can be improved in a number of ways. In particular, creating dual sheets that reside completely inside the volume is barely possible using our Freeform tool. Since existing high-quality hex meshes often contain dual sheets that are mostly parallel to the boundary surface, utilizing some other geometric information such as signed distance function seems promising for making our tool more intelligent and geometry-aware. Also, even though fully automated generation of dual sheets seems elusive, some kind of partial automation should be feasible where, starting from an existing configuration of dual sheets, the system would suggest a way to correct an existing sheet or to create a new sheet such that the sheet configuration becomes valid or better. Finally, visualization will be a key issue when dealing with intricate configurations of dual sheets inside the volume.
- Liu et al.'s method [LZC*18] supports boundary vertex topologies with boundary edges of valence 4 (cf. their paper's Fig. 3 bottom right) which seem to be of practical relevance. We will need to expand our set of reference polyhedra in order to support such topologies.
- Our system has been so far tested only against models of medium complexity. Our future work includes determining how our system scales to more complex examples, as well as conducting an extensive user study comparing our system against other commercial alternatives such as ANSYS [ANS18].

Acknowledgments

The author would like to thank Olga Sorkine-Hornung and Daniele Panozzo for their feedback at an early stage of this research, and the anonymous reviewers for their insights. This work was supported by JSPS KAKENHI Grant Number 15K15999.

References

[AFTR15] ARMSTRONG C. G., FOGG H. J., TIERNEY C. M., ROBINSON T. T.: Common themes in multi-block structured quad/hex mesh generation. In *Proc. IMR* (2015), pp. 70–82. 2

[ANS18] ANSYS: ANSYS Inc., 2018. Version 19.2, https://www.ansys.com/. 2, 11

[CBK12] CAMPEN M., BOMMES D., KOBBELT L.: Dual Loops Meshing: Quality Quad Layouts on Manifolds. *ACM Trans. Graph. 31*, 4 (2012), 110:1–110:11. 1, 2, 3, 6

[CK14] CAMPEN M., KOBBELT L.: Dual Strip Weaving: Interactive Design of Quad Layouts Using Elastica Strips. *ACM Trans. Graph. 33*, 6 (2014), 183:1–183:10. 1, 2, 6

[Eri14] ERICKSON J.: Efficiently Hex-Meshing Things with Topology. *Discrete Comput. Geom. 52*, 3 (2014), 427–449. 2

[ESCK16] EBKE H.-C., SCHMIDT P., CAMPEN M., KOBBELT L.: Interactively Controlled Quad Remeshing of High Resolution 3D Models. *ACM Trans. Graph. 35*, 6 (2016), 218:1–218:13. 2

[FXBH16] FANG X., XU W., BAO H., HUANG J.: All-hex Meshing Using Closed-form Induced Polycube. *ACM Trans. Graph. 35*, 4 (2016), 124:1–124:9. 2

[GDC15] GAO X., DENG Z., CHEN G.: Hexahedral Mesh Re-parameterization from Aligned Base-complex. *ACM Trans. Graph. 34*, 4 (2015), 142:1–142:10. 2, 3

[GPW*17] GAO X., PANOZZO D., WANG W., DENG Z., CHEN G.: Robust Structure Simplification for Hex Re-meshing. *ACM Trans. Graph. 36*, 6 (2017), 185:1–185:13. 2

[GSZ11] GREGSON J., SHEFFER A., ZHANG E.: All-Hex Mesh Generation via Volumetric PolyCube Deformation. *Comput. Graph. Forum 30*, 5 (2011), 1407–1416. 2

[HJS*14] HUANG J., JIANG T., SHI Z., TONG Y., BAO H., DESBRUN M.: ℓ1-Based Construction of Polycube Maps from Complex Shapes. *ACM Trans. Graph. 33*, 3 (2014), 25:1–25:11. 2

[HTWB11] HUANG J., TONG Y., WEI H., BAO H.: Boundary Aligned Smooth 3D Cross-frame Field. *ACM Trans. Graph. 30*, 6 (2011), 143:1–143:8. 2

[JHW*14] JIANG T., HUANG J., WANG Y., TONG Y., BAO H.: Frame Field Singularity Correction for Automatic Hexahedralization. *IEEE TVCG 20*, 8 (2014), 1189–1199. 2

[JP*18] JACOBSON A., PANOZZO D., ET AL.: libigl: A simple C++ geometry processing library, 2018. http://libigl.github.io/libigl/. 10

[JTPSH15] JAKOB W., TARINI M., PANOZZO D., SORKINE-HORNUNG O.: Instant Field-aligned Meshes. *ACM Trans. Graph. 34*, 6 (2015), 189:1–189:15. 2, 3

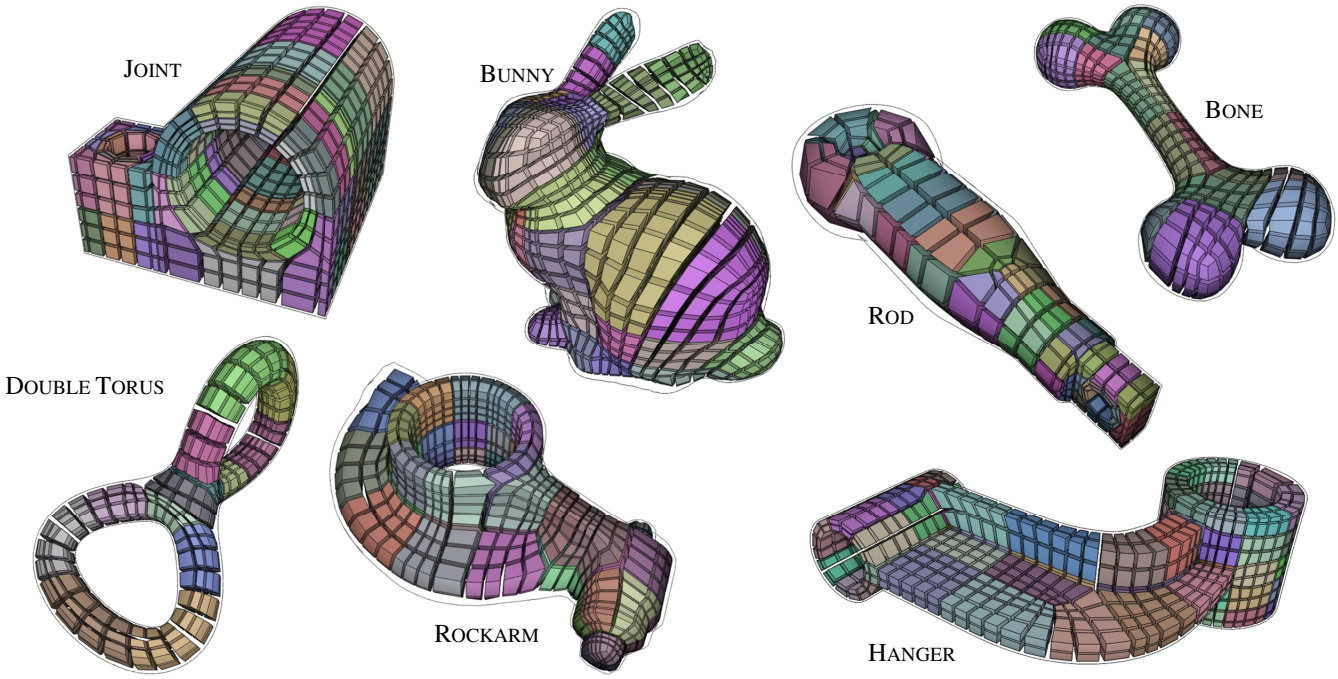


Figure 22: *Example hexahedralizations generated using our system.*

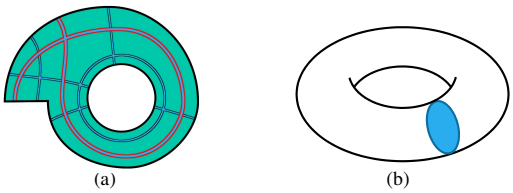


Figure 23: *Dual sheets that cannot be created using our system due to topological restrictions imposed by the use of implicit representation.*

[KBK13] KREMER M., BOMMES D., KOBBELT L.: OpenVolumeMesh – A Versatile Index-Based Data Structure for 3D Polytopal Complexes. In *Proc. IMR* (2013), pp. 531–548. 6

[KBLK13] KREMER M., BOMMES D., LIM I., KOBBELT L.: Advanced Automatic Hexahedral Mesh Generation from Surface Quad Meshes. In *Proc. IMR* (2013), pp. 147–164. 2

[LBK16] LYON M., BOMMES D., KOBBELT L.: HexEx: Robust Hexahedral Mesh Extraction. *ACM Trans. Graph. 35*, 4 (2016), 123:1–123:11. 2

[LLX*12] LI Y., LIU Y., XU W., WANG W., GUO B.: All-hex Meshing Using Singularity-restricted Field. *ACM Trans. Graph. 31*, 6 (2012), 177:1–177:11. 2

[LZC*18] LIU H., ZHANG P., CHIEN E., SOLOMON J., BOMMES D.: Singularity-constrained Octahedral Fields for Hexahedral Meshing. *ACM Trans. Graph. 37*, 4 (2018), 93:1–93:17. 2, 7, 8, 10, 11

[MGV11] MACÊDO I., GOIS J. P., VELHO L.: Hermite Radial Basis Functions Implicits. *Comput. Graph. Forum 30*, 1 (2011), 27–42. 3

[MH99] MÜLLER-HANNEMANN M.: Hexahedral Mesh Generation by Successive Dual Cycle Elimination. *Engineering with Computers 15*, 3 (1999), 269–279. 2

[MTP*15] MARCIAS G., TAKAYAMA K., PIETRONI N., PANOZZO D., SORKINE-HORNUNG O., PUPPO E., CIGNONI P.: Data-Driven Interactive Quadrangulation. *ACM Trans. Graph. 34*, 4 (2015), 65:1–65:10. 2

[NRP11] NIESER M., REITEBUCH U., POLTHIER K.: CubeCover – Parameterization of 3D Volumes. *Comput. Graph. Forum 30*, 5 (2011), 1397–1406. 2

[RSL16] RAY N., SOKOLOV D., LÉVY B.: Practical 3D Frame Field Generation. *ACM Trans. Graph. 35*, 6 (2016), 233:1–233:9. 2

[Si15] SI H.: TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator. *ACM Trans. Math. Softw. 41*, 2 (2015), 11:1–11:36. 3, 9

[SVB17] SOLOMON J., VAXMAN A., BOMMES D.: Boundary Element Octahedral Fields in Volumes. *ACM Trans. Graph. 36*, 3 (2017), 28:1–28:16. 2

[TBM96] TAUTGES T. J., BLACKER T., MITCHELL S. A.: The Whisker Weaving Algorithm: A Connectivity-Based Method for Constructing All–Hexahedral Finite Element Meshes. *Int. Journal of Numerical Methods in Engineering 39*, 19 (1996), 3327–3349. 2

[TPSHSH13] TAKAYAMA K., PANOZZO D., SORKINE-HORNUNG A., SORKINE-HORNUNG O.: Sketch-Based Generation and Editing of Quad Meshes. *ACM Trans. Graph. 32*, 4 (2013), 97:1–97:8. 2

[VSL16] VIERTEL R., STATEN M., LEDOUX F.: Analysis of Non-Meshable Automatically Generated Frame Fields. In *Proc. IMR* (2016). 11

[YZL15] YU W., ZHANG K., LI X.: Recent algorithms on automatic hexahedral mesh generation. In *Proc. ICCSE* (2015), pp. 697–702. 2