Volumetric Michell Trusses for Parametric Design & Fabrication: Supplemental Material

Rahul Arora
University of Toronto
Toronto, ON, Canada
arorar@dgp.toronto.edu

Alec Jacobson
University of Toronto
Toronto, ON, Canada
jacobson@cs.toronto.edu

Timothy R. Langlois
Adobe Research
Seattle, WA, USA
tlangloi@adobe.com

Yijiang Huang
Massachusetts Institute of Technology
Cambridge, MA, USA
yijiangh@mit.edu

Caitlin Mueller
Massachusetts Institute of Technology
Cambridge, MA, USA
caitlinm@mit.edu

Wojciech Matusik
MIT CSAIL
Cambridge, MA, USA
wojciech@csail.mit.edu

Ariel Shamir
The Interdisciplinary Center
Hertsliya, Israel
arik@idc.ac.il

Karan Singh
University of Toronto
Toronto, ON, Canada
karan@dgp.toronto.edu

David I.W. Levin
University of Toronto
Toronto, ON, Canada
diwlevin@cs.toronto.edu

1 SYMMETRIC FIELD REPRESENTATIONS

While we aim to solve a physical problem as described above, our algorithm is inspired by geometry processing work on field-aligned meshing. Numerous quad-meshing and hex-meshing methods (e.g., [Jakob et al. 2015; Nieser et al. 2011; Solomon et al. 2017]) aim to generate meshes whose elements follow certain "direction lines". Formally, these directions are represented using generalizations of classical vector fields to fields with rotational symmetry. For example, in 2D, a 4-rotationally symmetric (4-RoSy) field defines a set of four orthonormal vectors at each point on a manifold.

Solomon et al. [2017] refer to the corresponding 3D representation as an *octahedral field*. An octahedral field $O : \mathcal{M} \rightarrow \{\pm v_1, \pm v_2, \pm v_3\}$ assigns a set of six orthogonal unit vectors to each point, where $v_i \in \mathbb{R}^3$ are mutually orthogonal unit vectors. An equivalent representation, referred to as a *frame field* in literature [Nieser et al. 2011], can be defined by an arbitrary choice of three vectors forming a right-handed orthogonal frame (that is, $(u, v, w = u \times v) \in \mathbb{R}^{3\times3}$) from the octahedral field representation. The *frame* is then considered to be invariant to transformations under the chiral cubical symmetry group. We refer the interested reader to CubeCover [Nieser et al. 2011, Sections 2.1-2.2] and Liu et al. [2018, Section 4] for a more detailed discussion on these representations. The important takeaway for our work is that a frame field as defined above can be represented as a 3×3 orthonormal matrix, that is, a 3D-rotation matrix $R \in \mathrm{SO}(3)$. However, the equivalence under chiral cubical symmetry transformations implies that the matrix R is not unique, and 24 different rotation matrices represent the same frame.

To construct a Michell Truss structure, we want to align truss members to the eigenvectors of a stress tensor matrix. This being a real symmetric—and thus diagonalizable—matrix guarantees that an orthonormal 3D frame can be defined by its eigenvectors. Since eigenvectors are unsigned and have no canonical ordering, they require the same symmetric considerations of frame fields utilized in hex-meshing techniques. While this suggests that a frame-aligned meshing method could be adapted to work for computing stress-aligned frames, a careful examination of eigenspaces of matrices reveals that this is not always true. Consider a tensor $\sigma \in \mathbb{R}^{3\times3}$ with eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3$. The eigenspace associated with an eigenvalue λ of σ is defined as

$$\mathsf{E}(\sigma, \lambda) = \{\mathbf{v} : \sigma\mathbf{v} = \lambda\mathbf{v}\}. \tag{1}$$

For tensors with all distinct eigenvalues, the eigenspaces $E(\sigma, \lambda_i)$ are one-dimensional subspaces of $\mathbb{R}^3$ (lines), and the choice of a unit vector from each of the three distinct eigenspaces is guaranteed to give a unique frame (up to sign and ordering). However, at degenerate points of the stress tensor field [Hesselink et al. 1997; Palacios et al. 2017], that is, the points where the eigenvalues exhibit multiplicity[1], the eigenspaces do not form one-dimensional subspaces of $\mathbb{R}^3$. Instead, the eigenspace corresponding to the repeated eigenvalue forms a two-dimensional subspace when the multiplicity is two and spans all of $\mathbb{R}^3$ when multiplicity is 3. Thus, for degenerate points of the stress tensor field, the frame defined by the three orthogonal eigenvectors is not unique. As a consequence, we chose to work directly with the stress tensor field instead of converting to a frame field representation and adapting existing hex-meshing methods for our task. Figure 1 illustrates why this is the suitable choice for our problem.

2 MINIMIZERS OF EQUATION 3

Here, we prove that the non-truncated energy in Equation 3 (main document) admits the claimed set of extrema. To simplify the notation, we use M instead of σ_+ here. Also, let $\lambda_1 \geq \lambda_2 \geq \lambda_3 > 0$ be the eigenvalues of M. The M-norm of a vector $\mathbf{v}$ is given by

$$\begin{aligned}\|\mathbf{v}\|_M &= \sqrt{\mathbf{v}^T M\mathbf{v}} = \sqrt{\mathbf{v}^T Q\Lambda Q^T\mathbf{v}} \\ &= \sqrt{\mathbf{v}^T Q\Lambda^{1/2}\left(\Lambda^{1/2}\right)^T Q^T\mathbf{v}} = \sqrt{\left(\Lambda^{1/2}Q^T\mathbf{v}\right)^T\left(\Lambda^{1/2}Q^T\mathbf{v}\right)} \\ &= \|\Lambda^{1/2}Q^T\mathbf{v}\|_2,\end{aligned}$$

[1] Since the stress tensor is a diagonalizable matrix, the algebraic and geometric multiplicities are the same for any eigenvalue. Therefore, we simplify the discussion by simply using multiplicity everywhere.

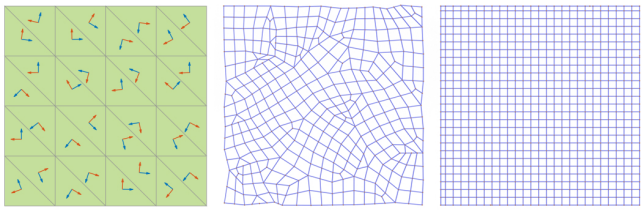


Figure 1: Degenerate tensors do not define unique (up to symmetry) frames. As this synthetic example with an everywhere degenerate tensor field shows, methods utilizing only the frame field defined by the eigenvectors, such as [Jakob et al. 2015], produce a highly distorted and non-smooth truss layout. Our method takes into account the actual eigenspaces and produces the optimal (constant) frame field resulting in a smooth truss.

by using the fact that $\Lambda^{1/2}$ is symmetric. Also, without loss of generality, let $Q = \mathbb{I}_3$, and therefore

$$E(R) = \sum_{i=1}^{3} \left\| \Lambda^{1/2} \mathbf{r}_i \right\|_2 = \sum_{i=1}^{3} \sqrt{\sum_{j=1}^{3} \lambda_j r_{ji}^2}$$

where r_{ij} is the $(i,j)^{\text{th}}$ element of R.

Consider the differential ΔE of E when changing the parameter R by an infinitesimal rotation ΔR. On SO(3), an infinitesimal step has the form

$$\Delta R = \mathbb{I}_3 + skew([x,y,z]) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}$$

where $x, y, z \to 0$.

The rotated matrix is given by $S = R\Delta R$. Now,

$$\begin{aligned} \Delta E &= E(S) - E(R) \\ &= \sum_{i=1}^{3} \sqrt{E_i(S)} - \sqrt{E_i(R)} \qquad \left(E_i(R) := \|\Lambda^{1/2} \mathbf{r}_i\|_2^2 \right) \\ &= \sum_{i=1}^{3} \Delta E_i . \qquad \left(\Delta E_i := \sqrt{E_i(S)} - \sqrt{E_i(R)} \right) \end{aligned}$$

$\Delta E = 0$ implies $E(R) = E(S)$. Squaring this and then using the Taylor series expansion of square-root while dropping second and higher-order terms involving x, y, and z gives us

$$\begin{aligned} \Delta E_1 \left(1 + \frac{E_2(R)}{E_1(R)} + \frac{E_3(R)}{E_1(R)} \right) + \Delta E_2 \left(1 + \frac{E_1(R)}{E_2(R)} + \frac{E_3(R)}{E_2(R)} \right) \\ + \Delta E_3 \left(1 + \frac{E_1(R)}{E_3(R)} + \frac{E_2(R)}{E_3(R)} \right) = 0 \end{aligned}$$

Since $E_i(R) > 0$, ΔE is zero iff all ΔE_i are independently zero.

Expanding ΔE_1 and dropping higher order terms of x, y, z, we get

$$\begin{aligned} \Big(E_1(R) + 2x(\lambda_2 - \lambda_3) r_{12} r_{13} + 2y(\lambda_3 - \lambda_1) r_{13} r_{11} \\ + 2z(\lambda_1 - \lambda_2) r_{11} r_{12} \Big)^{1/2} - \Big(E_1(R) \Big)^{1/2} = 0. \end{aligned}$$

Since x, y, z are arbitrary, this implies $(\lambda_2 - \lambda_3) r_{12} r_{13} = (\lambda_3 - \lambda_1) r_{13} r_{11} = (\lambda_1 - \lambda_2) r_{11} r_{12} = 0$. After analyzing E_2 and E_3 similarly, we get the following condition for characterizing the extrema of Equation 3 (main document).

$$\forall i,\ (\lambda_2 - \lambda_3) r_{i2} r_{i3} = (\lambda_3 - \lambda_1) r_{i3} r_{i1} = (\lambda_1 - \lambda_2) r_{i1} r_{i2} = 0. \quad (2)$$

If the eigenvalues are all equal, this condition always holds and thus every $R \in \text{SO}(3)$ is an extremum. If all eigenvalues are distinct, then the condition simplifies to

$$\forall i,\ \exists j,k\ (j \neq k), \text{s.t. } r_{ij}, r_{ik} = 0$$

This gives us 48 $O(3)$ matrices which have three entries ± 1 and the rest zeroes. Taking into account that the determinant of SO(3) matrices is $+1$, we get 24 solutions, which is exactly the set of matrices given by the chiral cubical symmetry group acting on $\mathbb{I}_3$.

The last case occurs when two eigenvalues are the same and the third is distinct. For the subcase $\lambda_1 = \lambda_2 \neq \lambda_3$, we get

$$\forall i\ \ (r_{i3} = 0 \text{ or } r_{i1} = r_{i2} = 0)\,. \quad (3)$$

The orthonormality of R implies that we must enforce the condition $r_{i3} = 0$ to two of its rows, and $r_{i1} = r_{i2} = 0$ to the remaining row. This gives us the set of matrices where $\mathbf{r}_3 = \pm \mathbf{e}_i$. That is, the 3^{rd} column of R is one of the coordinate bases, with sign. This represents the set of frames where one of the axes is always aligned with the $\pm$Z-axis, while the other two are any orthogonal pair in the XY-plane such that the right-hand the frame is right-handed. The other subcase when $\lambda_1 \neq \lambda_2 = \lambda_3$ is symmetric and gives the expected set of extrema—set of right-handed frames with one axis aligned with the $\pm$X-axis.

Lastly, to obtain the set of solutions for a general $Q \neq \mathbb{I}_3$, we can simply premultiply by Q. That is, if $\mathcal{R}_{\mathbb{I}}$ is the set of extrema when the eigenvector matrix is $\mathbb{I}_3$, then the set of extrema for the eigenvector matrix being Q is simply $\{QR : R \in \mathcal{R}_{\mathbb{I}}\}$.

□

3 TRUSS LAYOUT EXTRACTION DETAILS

Our stress fields are divergence-free in the interior of the domain since forces are only applied at the boundary. This implies that the principal stress lines must end at the boundary, and cannot end abruptly or form closed surfaces inside the domain. Since the first three steps of the algorithm ensure that the isocurves of $\widetilde{\phi}$ follow the principal stress lines, we make the assumption that they do not form internal closed surfaces as well.

Further, for ease of exposition, we will start by describing a 2D truss layout extraction algorithm. The boundary force only assumption in the 2D case implies that all isocurves of $\widetilde{\phi}$ do not form closed curves. Therefore, for tracing these isocurves, we start at their end points on the boundary, and trace until we hit the boundary again.

We use Γ_i to refer to the set of end-to-end integer isocurves of $\widetilde{\phi}_i$, and γ_i to refer to an arbitrary curve from this set. Note that $\widetilde{\phi}$ is a piecewise linear field stored at the vertices, and its value at arbitrary $\mathbf{x} \in \Omega$ can be found using Barycentric interpolation. However, Barycentric interpolation on a triangle is equivalent to linear interpolation along edges followed by interpolating across

Table 1: Notation for the truss layout extraction procedure described in §3.

Both 2D and 3D	
$\partial\Omega$	Boundary of the input domain
$\partial\mathcal{M}$	The triangle mesh bounding $\mathcal{M}$
$\mathcal{N}$	Points on the integer grid defined by $\widetilde{\phi}$ (the set of nodes of the truss layout)
$\mathcal{E}$	The set of elements of the truss layout
2D	
γ_i	An integer isocurve of $\widetilde{\phi}_i$
Γ_i	The set of all γ_i
$\mathcal{N}_{Ei}$	Intersection points b/w curves in Γ_i and all edges of $\mathcal{M}$
$\mathcal{N}_E$	Disjoint union of $\mathcal{N}_{E1}$ and $\mathcal{N}_{E2}$
$\mathcal{N}_e$	Points in $\mathcal{N}_E$ lying on a particular edge e
$\mathcal{N}_{f,\gamma_i}$	Points on the integer grid, lying on the intersection b/w γ_i and a particular face f
$\mathcal{N}_F$	Union of $\mathcal{N}_{f,\gamma_1}$ and $\mathcal{N}_{f,\gamma_2}$ over all faces of $\mathcal{M}$
$\mathcal{E}_i$	Set of all elements tracing integer isocurves of $\widetilde{\phi}_i$
3D	
$\mathcal{S}_i$	An integer isosurface of $\widetilde{\phi}_i$
γ_{ij}	An integer isocurve of $(\widetilde{\phi}_i, \widetilde{\phi}_j)$
$\mathcal{N}_F$	Points of intersections between all $\{\gamma_{ij}\}$ and faces of $\mathcal{M}$
$\mathcal{N}_E$	Points of intersections between all $\{\gamma_{ij}\}$ and edges of $\partial\mathcal{M}$

a line segment between two edges. We utilize this series of linear interpolations to trace out the integer isocurves of our parametrization.

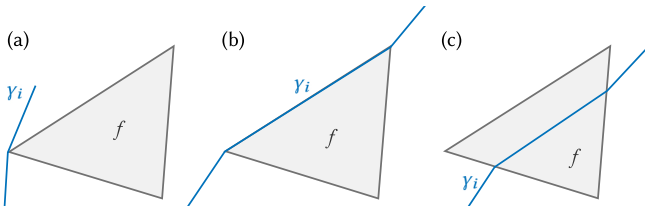


Figure 2: An integer isocurve γ_i can intersect a face f either on a single vertex (a), on an edge (b), or go through its interior (c). We perturb the parametrization by an infinitesimal amount to eliminate the first two cases.

In order to make this approach work, we require that an isocurve and a face intersect in exactly two points (or do not intersect at all). The only excluded cases are when an isocurve just touches a face at a single vertex, or is aligned with one of the edges (Figure 2). While we never encountered these cases with our paramterizations, we can easily eliminate the theoretical possibility as well. We find the parameter values on vertices which are close to integers up to machine precision, and translate them by $-\epsilon$, where ϵ is a small positive number (we choose 10^{-7}). If the vertex also has the minimum parameter value in its 1-ring neighbourhood, we translate by $+\epsilon$ instead, ensuring that we do not remove part of an integer isocurve. This ensures that no γ_i passes through a vertex, eliminating both the problematic cases.

Starting from the parameter values stored at the vertices, we use linear interpolation along edges to find the intersections of curves from Γ_1 and Γ_2 with the edges to form the sets of nodes $\mathcal{N}_{E1}$ and $\mathcal{N}_{E2}$, respectively (Figure 3a). These nodes are then used to find nodes in the interior of faces, and their connectivity, as described below.

Consider a face f and an isocurve $\gamma_1 \in \Gamma_1$ intersecting with it. Let $\mathbf{x}_0, \mathbf{x}_1 \in \mathcal{N}_{E1}$ be the end points of the line of intersection. We linearly interpolate the values of $\widetilde{\phi}_1$ on $\mathbf{x}_0$ and $\mathbf{x}_1$ to find the points of intersection of this isoline with all $\gamma_2 \in \Gamma_2$:

$$\mathcal{N}_{f,\gamma_1} = \left\{ \mathbf{y} \in f \cap \gamma_1 \mid \widetilde{\phi}_2(\mathbf{y}) \in \mathbb{Z} \cap \left(\widetilde{\phi}_2(\mathbf{x}_0), \widetilde{\phi}_2(\mathbf{x}_1) \right) \right\}, \quad (4)$$

where $\widetilde{\phi}_2(\mathbf{x}_0) \leq \widetilde{\phi}_2(\mathbf{x}_1)$ *wlog*. $\mathcal{N}_{f,\gamma_1}$ is then sorted by $\widetilde{\phi}_2$, the two extrema are connected to $\mathbf{x}_0$ and $\mathbf{x}_1$, and consecutive points in the ordered set are connected to each other. Doing this for all admissible pairs (f, γ_1) gives the set $\mathcal{N}_F$ of nodes lying on the intersections between all pairs (γ_1, γ_2), and the set of elements $\mathcal{E}_1$ tracing all $\gamma_1 \in \Gamma_1$ (Figure 3b–c).

Then, for each pair of intersecting face and γ_2, we search for the nodes among $\mathcal{N}_F$ lying on the intersection. The nodes lying on each γ_2 are then connected to form the set of elements $\mathcal{E}_2$ (Figure 3d). Define $\mathcal{N}_E$ to be the disjoint union of $\mathcal{N}_{E1}$ and $\mathcal{N}_{E2}$, and $\mathcal{N}_{\partial\Omega}$ to be the subset of $\mathcal{N}_E$ restricted to nodes lying on the boundary. In the final step of the algorithm, we insert all nodes from $\mathcal{N}_{\partial\Omega}$ into a queue and trace out the integer isocurves emanating from them, going through the faces it intersects until we hit the boundary again. For each traced curve, we remove its endpoints from the queue.

3.0.1 3D Truss Layouts. In 3D we use $\mathcal{S}_i$ to denote an arbitrary end-to-end integer isosurface of $\widetilde{\phi}_i$, and γ_{ij} to denote an arbitrary end-to-end curve where both $\widetilde{\phi}_i$ and $\widetilde{\phi}_j$ are constant integers. After perturbing the parametrization, we compute the points of intersection of isosurfaces of each of the three parameters with the edges. Then, we compute the intersection points of each γ_{ij} with all the faces of the mesh. Finally, we linearly interpolate $\widetilde{\phi}_k (k \neq i, j)$ along these isolines in each tet, and compute the intersections with all $\mathcal{S}_k$ to find the elements of the truss. Note that while the perturbation does not guarantee that every γ_{ij} passes through the interior of all tets—it may just touch it at a face—we never encountered this case in practice.

An additional complication is caused by the presence of closed isocurves. Note that two open surfaces $\mathcal{S}_i, \mathcal{S}_j$ can intersect in a closed curve, and thus our assumption of no closed isosurfaces does not guarantee that all isocurves are also open. To extract these curves, we simply search for *degree-deficient* nodes in $\mathcal{N}$, that is, nodes with degree below 6. We expect all integer-mapped nodes to have a degree of 6, since each of those lies on three isocurves, one for each $\{i, j\} \in \binom{3}{2}$. We start by arbitrarily choosing a node (say $\mathbf{x}$)

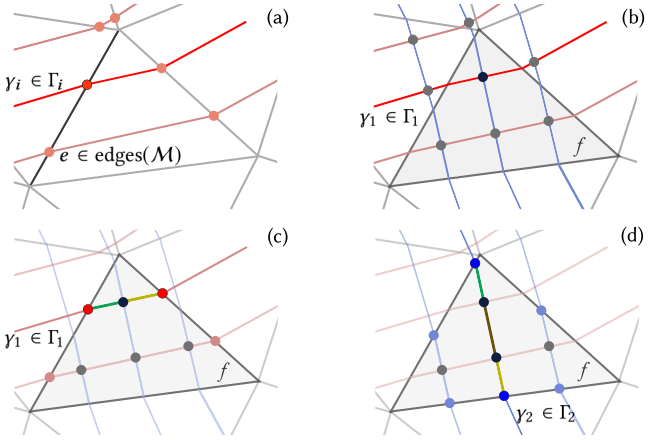
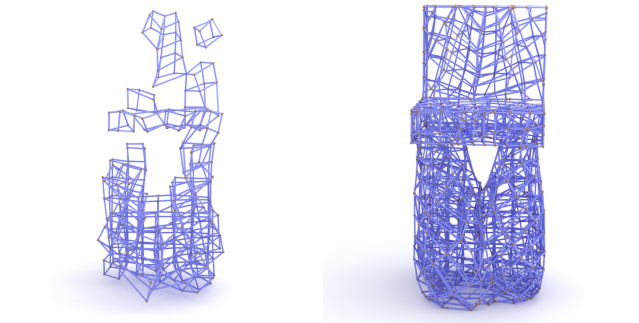


Figure 3: In 2D, the truss extraction process begins by finding all points of intersections between integer isocurves of $\widetilde{\phi}$ and edges of the input mesh (a). Then, for each face of the mesh f, and an integer isocurve of $\widetilde{\phi}_1$ intersecting with it, points mapped to the integer grid are located (b). All such points form the set of nodes $\mathcal{N}$ for the truss. Finally, the linear section of each $\widetilde{\phi}_1$ isocurve is cut along these points to form the elements $\mathcal{E}_1$ for the output truss (c), followed by a similar discretization and tracing process for $\widetilde{\phi}_2$ isolines to form $\mathcal{E}_2$ (d). The union of $\mathcal{E}_1$ and $\mathcal{E}_2$ is the set of elements $\mathcal{E}$ of the truss.

from a list of all degree-deficient nodes, and trace out the missing isocurve(s) until we hit **x** again. During this process, we update the degree of all nodes we encounter. The selection and tracing steps are then repeated until no degree-deficient nodes are left.



HexEx [Lyon et al. 2016] **Ours**

Figure 4: Hex-mesh extraction methods assume a boundary-aligned parametrization, and therefore, fail to extract truss members lying on or touching the boundary (left). Our truss extraction algorithm manages to extract the complete structure, including the boundary (right).

3.0.2 Handling the boundary. As noted earlier, existing work on hex-meshing assumes that the parametrization is defined such that for all points on the domain boundary, at least one of the parameter values is an integer. Since this is not true for our parametrizations, we also have to include additional nodes on the boundary, along with elements connecting these to each other and to the internal nodes. Figure 4 shows how HexEx [Lyon et al. 2016]—a state-of-the-art mesh extraction method—fails to extract the boundary for our example.

Note that the procedure described above already traces out the nodes on the boundary, as well the elements which touch it. To find the elements lying *on* the boundary, we need to trace the intersection of each $\mathcal{S}_i$ with the boundary $\partial\mathcal{M}$, which comes down to performing the full 2D truss extraction procedure for each pair of parameters $\{i, j\} \in \binom{3}{2}$ on the triangle mesh $\partial\mathcal{M}$. A small change is that these curves are closed, and so we have to keep track of the (arbitrarily chosen) initial point for each curve and trace the isocurve until we hit the initial point again.

3.0.3 Simplification. We contract elements of the extracted truss layout until we have no nodes from $\mathcal{N}_F$ left, except those on the boundary. This gives us a graph similar to that in Equations 12–13 in the main document, but with additional nodes and elements on the boundary.

We can simplify the boundary elements as well by contracting elements containing the nodes in $\mathcal{N}_E$. We perform both these steps for all our results, but we do not remove boundary nodes which lie on feature edges. These features are currently selected using dihedral angles with a selection threshold of $\cos^{-1}(0.9)$ (approx. 25°), but one could easily plug in user-provided features. Finally, we trace these feature edges as well to preserve surface details.

4 GENERALITY OF THE ALGORITHM

To demonstrate the generality of our approach, we investigate the effect of changing boundary conditions and apply our method to objects with non-trivial topologies.

4.1 Effect of Boundary Conditions

First we explore the effect of traction boundary conditions on algorithm output. Figure 5 shows three identical bars subject to (resp.) shearing, compression, and torsion loads. Note how the resulting trusses adapt to these different boundary conditions. We also show the effect of varying the traction boundary conditions for a more complex chair shape. The two loading conditions correspond to a sitting pose and a rocking chair pose. That is, a downward force is applied on the seat in the former case, while the latter adds a compressive force on the backrest as well.

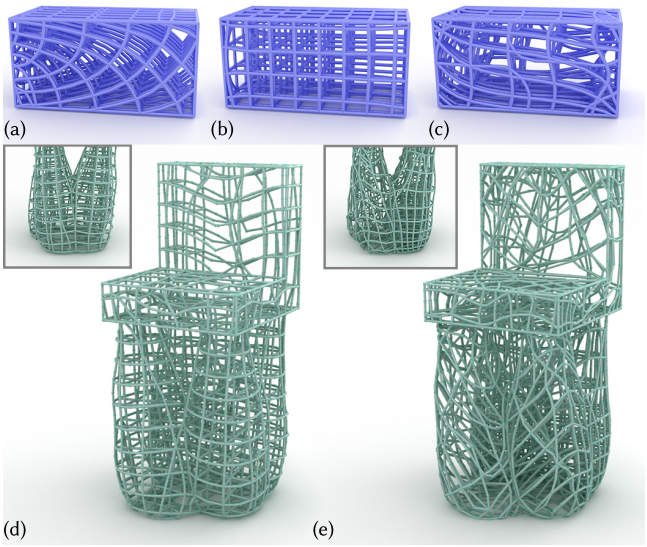


Figure 5: Effect of applying different boundary conditions on the same domain. Top row: The left face of the beam is fixed while the right face is subject to downward pulling (a), tension (b), and torsion (c) along the right face. Bottom row: the chair is optimized for a purely downward sitting load (d) and for the sitting load combined with a force pushing into the backrest (e). Notice the "bunched up" truss elements along the top of the backrest as well as directly below it in the chair's feet (inset) when the backrest force is added.

4.2 Effect of Complex Geometry

We also explore the expressive power of Michell Trusses by building trusses over a variety of complex shapes. Lacking a numerical method for constructing Michell Trusses for general 3D domains, prior work [Zegard and Paulino 2015, 2016] tends to construct Michell Trusses over geometrically simple and symmetric domains. Our method allows the novel capability of creating Michell Trusses over arbitrary 3D shapes. Construction of Michell Trusses over such interesting, geometrically complex input domains is demonstrated throughout the paper. In particular, we show trusses optimized over high-genus input domains in Figure 6, while Figure 1 in the main document shows a Michell Truss over a shape with numerous large concave regions. Quantitative measures aside, these results also show that our volumetric Michell Trusses preserve the "aesthetic" of the input shape even when it exhibits complex geometric features. We believe that this property makes our trusses well suited for design applications—particularly in architectural design where preservation of visual quality and overall shape is especially important [Dapogny et al. 2017]. In the next section, we discuss parametric post-processing operations enabled by our method which further this goal of integrating visual design and structural optimization.

5 ADDITIONAL EDITING OPERATIONS

We described four editing operations utilizing the global parametrization in the main document. Here, we detail two additional operations.

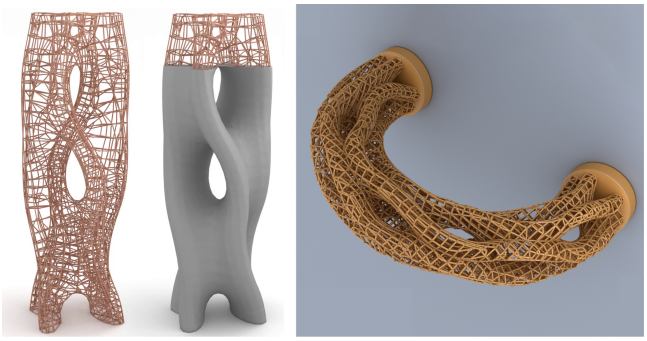


Figure 6: Our method works for arbitrary 3D shapes—independent of the homotopy class of the input mesh—as illustrated by these high genus examples. (Left) a genus-3 pillar is constructed by using our Michell Truss as the rebar skeleton before concrete is poured into the pillar shell. (Right) a high-genus input mesh is used to create a truss sculpture using metal bars. Input shapes created using Carlo Séquin's Scuplture Generator (https://people.eecs.berkeley.edu/~sequin/SCULPTS/scherk.html).

5.1 Vertex Snapping

We also allow users to enforce that a few selected vertices of the input mesh always have truss nodes lying on them.

The user specifies a subset of fixed vertices $V_0 \subset V$ and the parametrization is updated by solving the least-squares minimization with an appropriate constraint applied:

$$\phi^* = \underset{\phi}{\operatorname{argmin}} \sum_{i=1}^{3} \left\| \begin{bmatrix} G_x & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & G_y & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & G_z \end{bmatrix} \left(\widetilde{\phi}_i - \phi_i \right) \right\|_2^2, \qquad (5)$$
$$\text{s.t. } \forall v \in V_0 \;\; \phi_i(v) = \text{round}\left(\widetilde{\phi}_i(v) \right).$$

The truss extraction is then performed on the parametrization ϕ^* instead of $\widetilde{\phi}$. This has the effect of *snapping* the parameter grid to the selected vertices, while still closely following the principal stress lines elsewhere. An example use case is snapping to sharp visual features, resulting in aesthetically pleasing results, as shown in Figure 7. Again, this useful user interaction is only possible because of the global parametrization created by our method. Another potential use of this interaction could be for building interconnected trusses by optimizing over multiple objects independently and forcing the truss layouts to snap to the desired connection points between objects.

5.2 Curve Removal

Another advantage of maintaining a whole object parametrization is that it allows semantically meaningful curve selection. This results from the fact that a single, object spanning curve is mapped to an integer coordinate curve. We use this to enable curve deletion. Figure 8 shows an example of removing a spurious boundary curve from a design for aesthetic purposes. As with the preceding two operations, this useful manipulation is enabled by our algorithm and simply not available with output produced by other approaches.

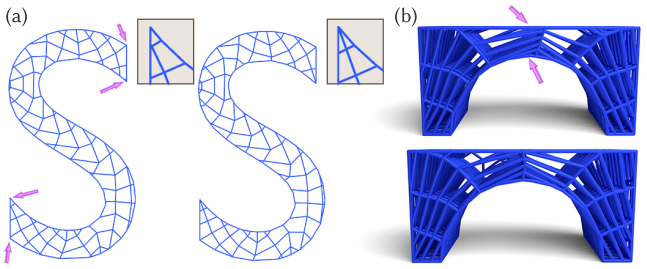


Figure 7: The user can specify certain vertices of the input mesh as constraints (violet circles), forcing a truss node to lie on each constrained vertex. The user can use these constraints, for example, to snap the truss to sharp vertices (a) or for symmetry (b).

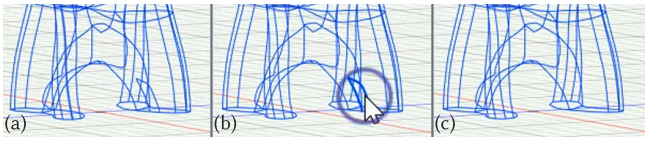


Figure 8: The existence of a global parametrization allows a user to quickly select an offending curve in a truss (a-b) and delete it to obtain the desired result (c).

6 PERFORMANCE

Table 2 concisely lists all our test geometries along with performance statistics. The reported performance statistics are for an Intel Xeon E5-2637 (3.5GHz) workstation utilizing 64GB of memory. Note that most of our code is written in MATLAB and is not optimized for speed.

7 IMPORTANCE OF STRESS-ALIGNMENT OBJECTIVE

Figure 9 shows the importance of our stress-alignment objective by comparing our result to a stress field-agnostic hex-meshing technique and to the trivial axis-aligned parametrization.

REFERENCES

Charles Dapogny, Alexis Faure, Georgios Michailidis, Grégoire Allaire, Agnes Couvelas, and Rafael Estevez. 2017. Geometric constraints for shape and topology optimization in architectural design. *Computational Mechanics* 59, 6 (01 Jun 2017), 933–965. https://doi.org/10.1007/s00466-017-1383-6

Xifeng Gao, Wenzel Jakob, Marco Tarini, and Daniele Panozzo. 2017. Robust Hex-dominant Mesh Generation Using Field-guided Polyhedral Agglomeration. *ACM Trans. Graph.* 36, 4, Article 114 (July 2017), 13 pages. https://doi.org/10.1145/3072959.3073676

L. Hesselink, Y. Levy, and Y. Lavin. 1997. The topology of symmetric, second-order 3D tensor fields. *IEEE Transactions on Visualization and Computer Graphics* 3, 1 (Jan 1997), 1–11. https://doi.org/10.1109/2945.582332

Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. 2015. Instant Field-Aligned Meshes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH ASIA)* 34, 6 (Nov. 2015). https://doi.org/10.1145/2816795.2818078

Heng Liu, Paul Zhang, Edward Chien, Justin Solomon, and David Bommes. 2018. Singularity-constrained Octahedral Fields for Hexahedral Meshing. *ACM Trans. Graph.* 37, 4, Article 93 (July 2018), 17 pages. https://doi.org/10.1145/3197517.3201344

Max Lyon, David Bommes, and Leif Kobbelt. 2016. HexEx: Robust Hexahedral Mesh Extraction. *ACM Trans. Graph.* 35, 4, Article 123 (July 2016), 11 pages. https://doi.org/10.1145/2897824.2925976

M. Nieser, U. Reitebuch, and K. Polthier. 2011. CubeCover—Parameterization of 3D Volumes. *Computer Graphics Forum* 30, 5 (2011), 1397–1406. https://doi.org/10.1111/j.1467-8659.2011.02014.x

Jonathan Palacios, Lawrence Roy, Prashant Kumar, Chen-Yuan Hsu, Weikai Chen, Chongyang Ma, Li-Yi Wei, and Eugene Zhang. 2017. Tensor Field Design in Volumes. *ACM Trans. Graph.* 36, 6, Article 188 (Nov. 2017), 15 pages. https://doi.org/10.1145/3130800.3130844

Justin Solomon, Amir Vaxman, and David Bommes. 2017. Boundary Element Octahedral Fields in Volumes. *ACM Trans. Graph.* 36, 3, Article 28 (May 2017), 16 pages. https://doi.org/10.1145/3065254

Tomás Zegard and Glaucio H. Paulino. 2015. GRAND3 – Ground Structure Based Topology Optimization for Arbitrary 3D Domains Using MATLAB. *Struct. Multidiscip. Optim.* 52, 6 (Dec. 2015), 1161–1184. https://doi.org/10.1007/s00158-015-1284-2

Tomás Zegard and Glaucio H. Paulino. 2016. Bridging topology optimization and additive manufacturing. *Structural and Multidisciplinary Optimization* 53, 1 (01 Jan 2016), 175–192. https://doi.org/10.1007/s00158-015-1274-4

Table 2: Performance results for all the 3D testcases. All reported runtimes rounded off to the nearest 1/10th of a second.

Example	# Vertices	# Tets	Resolution (ρ)	Sim (s)	Frames (s)	Tex (s)	Extract (s)	Total (s)
Curved bridge	1240	3843	48	11.1	35.8	0.1	241.5	288.5
Mars lander (upper leg)	2359	10327	24	49.2	30.1	0.6	164.3	244.2
Satellite antenna arm	3244	13367	32	44.3	49.3	0.9	549.7	644.2
Holey pillar	4117	15071	32	45.1	69.0	1.2	344.3	459.6
Cantilever beam	3457	16704	24	7.8	42.6	1.1	393.4	444.9
Bar under torsion	3457	16704	32	5.9	43.0	1.1	1086.0	1136.0
Bar under tension	3457	16704	32	22.0	32.7	1.0	879.4	935.1
Simple bridge	4382	19011	32	65.5	275.0	1.6	572.0	914.1
Mars Lander (lower leg)	4599	21825	24	101.1	69.0	1.9	190.0	362.0
Pavilion	7308	23512	48	101.9	140.5	2.9	478.1	723.4
Bookcase	6261	23962	32	83.9	96.6	2.5	372.4	555.4
Mars Lander (body)	6564	24945	32	123.3	169.0	2.6	757.2	1052.1
Arched bridge	6457	31164	32	66.2	183.1	3.4	918.4	1171.1
Quadcopter frame	7121	32826	24	37.6	127	4.2	393.2	562
Helicopter top pylon	8779	37797	32	190.9	161.9	5.3	357.2	715.3
Chair (sitting load)	9801	46187	32	21.1	177.4	38.3	1083.9	1320.7
Chair (rocking load)	9801	46187	32	85.7	165.1	7.1	1168.1	1426
Climbing hold	13753	68773	24	39.8	229.4	15.5	541.1	825.8
Holey sculpture	18016	79270	48	432.0	323.2	103.3	1282.3	2140.8
Jet engine bracket	28041	131796	32	700.9	4770.3	74.6	1229.3	6775.1

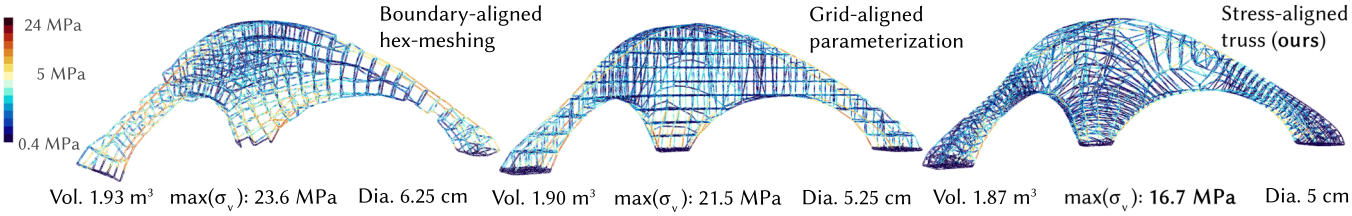


Figure 9: To verify the importance of our stress alignment objective, we created two variants of the bridge truss using a boundary-aligned hex-mesh [Gao et al. 2017] and by tracing the isolines of a trivial world-aligned parametrization. Von Mises stress computed under identical loads (lower is better) show that our stress-aligned truss significantly outperforms both.