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 \( O : M \to \{\pm v_1, \pm v_2, \pm v_3\} \) 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, \((u, v, w = u \times v) \in \mathbb{R}^3\times\mathbb{R}^3\)) 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 \(3 \times 3\) orthonormal matrix, that is, a 3D-rotation matrix \( R \in SO(3)\). 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 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 \( \lambda \) of \( \sigma \) is defined as

\[
E(\sigma, \lambda) = \{v : \sigma v = \lambda v\}.
\]

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 \( \sigma \) here. Also, let \( \lambda_1 \geq \lambda_2 \geq \lambda_3 > 0 \) be the eigenvalues of \( M \). The \( M \)-norm of a vector \( v \) is given by

\[
\|v\|_M = \sqrt{v^T MV} = \sqrt{v^T Q \Lambda Q^T v} = \sqrt{v^T Q \Lambda^{1/2} (\Lambda^{1/2} Q^T v)^T (\Lambda^{1/2} Q^T v)} = \|\Lambda^{1/2} Q^T v\|_2.
\]

\(^1\)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.
by using the fact that $A^{1/2}$ is symmetric. Also, without loss of
generality, let $Q = I_3$, and therefore

$$E(R) = \frac{3}{i=1} \left\| A^{1/2} r_i \right\|_2 = \frac{3}{i=1} \left( \sum_{j=1}^3 r_{ij}^2 \right)$$

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

Consider the differential $\Delta E$ when changing the parameter
$R$ by using the fact that $\Lambda$ is a piecewise linear field stored at the vertices, and its value
when the eigenvector matrix is $\phi$ is simply

$$\Delta E = E(S) - E(R)$$

$$\Delta E = \sum_{i=1}^3 \sqrt{E_i(S)} - \sqrt{E_i(R)}$$

We use $E_i = \lambda_i$ for $i = 1, 2, 3$.

The Taylor series expansion of square-root while dropping second and
higher-order terms involving $x, y,$ and $z$ gives us

$$\Delta E_i \left( 1 + \frac{E_{2i}(R)}{E_{i}(R)} + \frac{E_{3i}(R)}{E_{i}(R)} \right) + \Delta E_2 \left( 1 + \frac{E_{1i}(R)}{E_{i}(R)} + \frac{E_{4i}(R)}{E_{i}(R)} \right) \cdots = 0$$

$$\Delta E_i \left( 1 + \frac{E_{1i}(R)}{E_{i}(R)} + \frac{E_{2i}(R)}{E_{i}(R)} \right) \cdots = 0$$

Since $E_i(R) > 0$, $\Delta E$ is zero if all $\Delta E_i$ are independently zero.

Expanding $\Delta E_i$ and dropping higher order terms of $x, y, z$, we get

$$\left( E_i(R) + 2(x_0^3 - \lambda_3 r_{12} r_{13} + 2y_3^3 - \lambda_4 r_{13} r_{11} + 2z_3^3 - \lambda_2 r_{11} r_{12} \right) \frac{1}{2} \left( E_i(R) \right)^{1/2} = 0.$$
Table 1: Notation for the truss layout extraction procedure described in §3.

<table>
<thead>
<tr>
<th>Both 2D and 3D</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \partial \Omega )</td>
</tr>
<tr>
<td>( \partial M )</td>
</tr>
<tr>
<td>( N )</td>
</tr>
<tr>
<td>( \mathcal{E}_i )</td>
</tr>
</tbody>
</table>

2D

| \( Y_i \) | An integer isocurve of \( \bar{\phi}_i \) |
| \( \Gamma_i \) | The set of all \( Y_i \) |
| \( N_{Ei} \) | Intersection points b/w curves in \( \Gamma_i \) and all edges of \( M \) |
| \( N_E \) | Disjoint union of \( N_{E1} \) and \( N_{E2} \) |
| \( N_e \) | Points in \( N_E \) lying on a particular edge \( e \) |
| \( N_{f,Y_i} \) | Points on the integer grid, lying on the intersection b/w \( Y_i \) and a particular face \( f \) |
| \( N_F \) | Union of \( N_{f,Y_i} \) and \( N_{f,Y_j} \) over all faces of \( M \) |
| \( \mathcal{E}_i \) | Set of all elements tracing integer isocurves of \( \bar{\phi}_i \) |

3D

| \( S_i \) | An integer isosurface of \( \bar{\phi}_i \) |
| \( Y_{ij} \) | An integer isocurve of \( \bar{\phi}_i, \bar{\phi}_j \) |
| \( N_F \) | Points of intersection between all \( \{Y_{ij}\} \) and faces of \( M \) |
| \( N_E \) | Points of intersections between all \( \{Y_{ij}\} \) and edges of \( \partial M \) |

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 \( Y_i \) 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.](image)

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 parameterizations, 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 \( \epsilon \) 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 \( Y_i \) 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 \( \Gamma_1 \) and \( \Gamma_2 \) with the edges to form the sets of nodes \( N_{E1} \) and \( N_{E2} \), 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 \( Y_1 \in \Gamma_1 \) intersecting with it. Let \( x_0, x_1 \in N_{E1} \) be the end points of the line of intersection. We linearly interpolate the values of \( \bar{\phi}_1 \) on \( x_0 \) and \( x_1 \) to find the points of intersection of this isoline with all \( Y_2 \in \Gamma_2 \):

\[
N_{f,Y_1} = \left\{ y \in f \cap Y_1 \mid \bar{\phi}_1(y) \in \mathbb{Z} \cap \left( \bar{\phi}_2(x_0), \bar{\phi}_2(x_1) \right) \right\},
\]

where \( \bar{\phi}_2(x_0) \leq \bar{\phi}_2(x_1) \) wlog. \( N_{f,Y_1} \) is then sorted by \( \bar{\phi}_2 \), the two extrema are connected to \( x_0 \) and \( x_1 \), and consecutive points in the ordered set are connected to each other. Doing this for all admissible pairs \( (f, Y_1) \) gives the set \( N_F \) of nodes lying on the intersections between all pairs \( (Y_1, Y_2) \), and the set of elements \( \mathcal{E}_1 \) tracing all \( Y_1 \in \Gamma_1 \) (Figure 3b–c).

Then, for each pair of intersecting face \( f \) and \( Y_2 \), we search for the nodes among \( N_F \) lying on the intersection. The nodes lying on each \( Y_2 \) are then connected to form the set of elements \( \mathcal{E}_2 \) (Figure 3d). Define \( N_E \) to be the disjoint union of \( N_{E1} \) and \( N_{E2} \), and \( N_{E0} \) to be the subset of \( N_E \) restricted to nodes lying on the boundary. In the final step of the algorithm, we insert all nodes from \( N_{E0} \) 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 \( S_i \) to denote an arbitrary end-to-end integer isosurface of \( \bar{\phi}_i \), and \( Y_{ij} \) to denote an arbitrary end-to-end curve where both \( \bar{\phi}_i \) and \( \bar{\phi}_j \) 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 \( Y_{ij} \) with all the faces of the mesh. Finally, we linearly interpolate \( \bar{\phi}_k(k \neq i, j) \) along these isolines in each tet, and compute the intersections with all \( S_k \) to find the elements of the truss. Note that while the perturbation does not guarantee that every \( Y_{ij} \) 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 isosurfaces. Note that two open surfaces \( S_i, S_j \) can intersect in a closed curve, and thus our assumption of no closed isosurfaces does not guarantee that all isosurfaces are also open. To extract these curves, we simply search for degree-deficient nodes in \( N \), 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 \( \{i, j\} \in \binom{\mathbb{Z}}{2} \). We start by arbitrarily choosing a node (say \( x \))
Arora, R. et al

Figure 3: In 2D, the truss extraction process begins by finding all points of intersections between integer isocurves of $\phi$ and edges of the input mesh (a). Then, for each face of the mesh $f_i$ and an integer isocurve of $\phi_1$ intersecting with it, points mapped to the integer grid are located (b). All such points form the set of nodes $N$ for the truss. Finally, the linear section of each $\phi_1$ isocurve is cut along these points to form the elements $E_1$ for the output truss (c), followed by a similar discretization and tracing process for $\phi_2$ isolines to form $E_2$ (d). The union of $E_1$ and $E_2$ is the set of elements $E$ 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.

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 as the elements which touch it. To find the elements lying on the boundary, we need to trace the intersection of each $S_j$ with the boundary $\partial M$, which comes down to performing the full 2D truss extraction procedure for each pair of parameters $(i, j) \in \binom{\ell}{2}$ on the triangle mesh $\partial M$. 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 $N_F$ 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 $N_E$. 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. $25^\circ$), 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.

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).
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.

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:

\[
\phi^* = \arg\min_\phi \sum_{i=1}^3 \left\| \begin{bmatrix} G_x & 0 & 0 \\ 0 & G_y & 0 \\ 0 & 0 & G_z \end{bmatrix} \left( \hat{\phi}_i - \phi_i \right) \right\|_2^2, \quad (5)
\]

subject to \(\forall v \in V_0 \, \phi_i(v) = \text{round} \left( \hat{\phi}_i(v) \right)\).

The truss extraction is then performed on the parametrization \(\phi^*\) instead of \(\hat{\phi}\). 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 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.

Figure 6: Our method works for arbitrary 3D shapes—indepedent 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 re-bar 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 Sculpture Generator (https://people.eecs.berkeley.edu/~sequin/SCULPTS/scherk.html).
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


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

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

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.