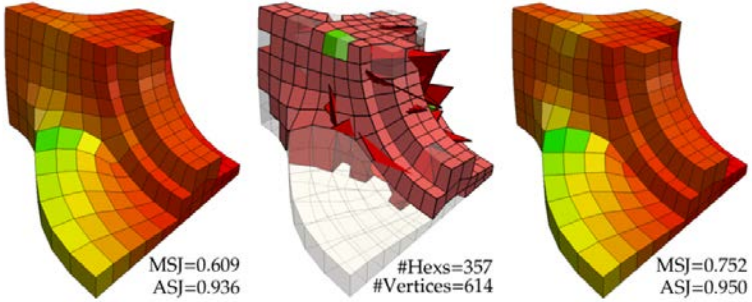
Hexahedral Mesh Quality Improvement via Edge-Angle Optimization

Kaoji Xu[1], Xifeng Gao[2], Guoning Chen[1]

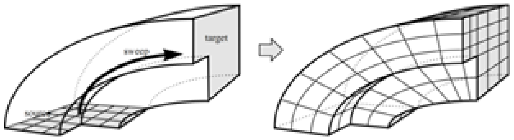
[1]University of Houston
[2]New York University



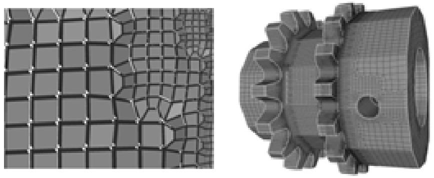
UNIVERSITY of HOUSTON

Existing Hex-Meshing Techniques

Sweeping/Octree



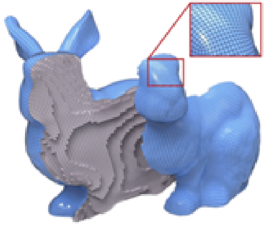
[Owen, et al. 1998]



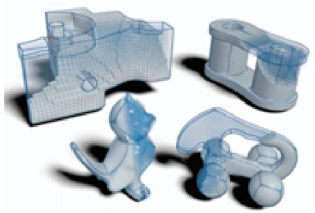
[Maréchal, et al. 2009]

Frame Field

[Nieser, et al. 2011]

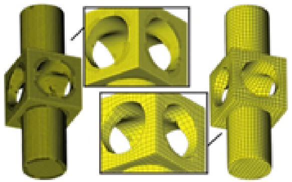


[Li, et al. 2012]

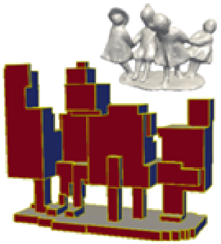


[Jiang, et al. 2013]

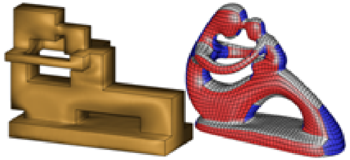
Polycube



[Gregson, et al. 2011]



[Huang, et al. 2014]



[Livesu, et al. 2013]

Existing Hex-Meshing Techniques

Sweeping/Octree

[Owen, et al. 1998]

[Maréchal, et al. 2009]

The initial hex-meshes generated by these methods are usually not optimal and may not meet the quality requirements of the target applications.

Polycube

[Gregson, et al. 2011]

[Huang, et al. 2014]

[Livesu, et al. 2013]

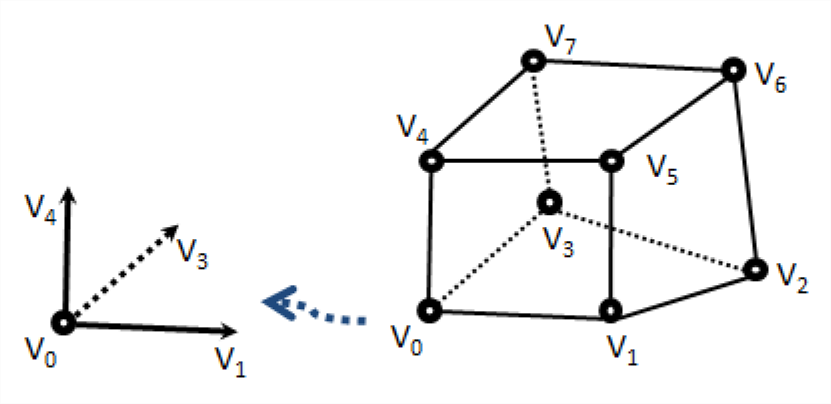
Jacobian Metrics for Hex-mesh Quality

$$\det(A_0) = \overrightarrow{v_1v_0} \cdot (\overrightarrow{v_3v_0} \times \overrightarrow{v_4v_0})$$

$$J_{jacobian} = \min\{\det(A_i), i = 0,1,2\cdots,7\}$$

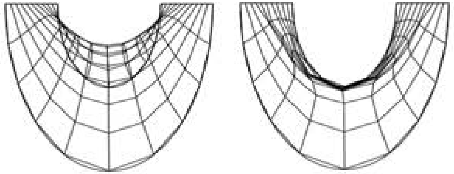
$$\det(\hat{A}_0) = \frac{\overrightarrow{v_1v_0}}{\|\overrightarrow{v_1v_0}\|} \cdot \left(\frac{\overrightarrow{v_3v_0}}{\|\overrightarrow{v_3v_0}\|} \times \frac{\overrightarrow{v_4v_0}}{\|\overrightarrow{v_4v_0}\|}\right)$$

$$S.J_{scaled\_jacobian} = \min\{\det(\hat{A}_i), i = 0,1,2\cdots,7\}$$

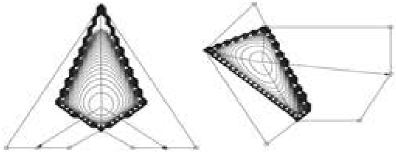


Existing Hex-Meshing Optimization Techniques

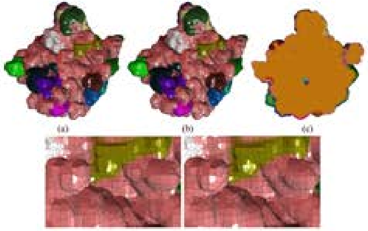
Geometric Optimization



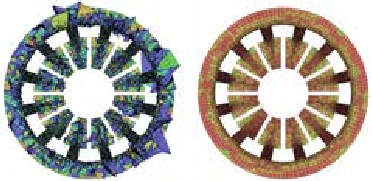
[Knupp, et al. 2000]



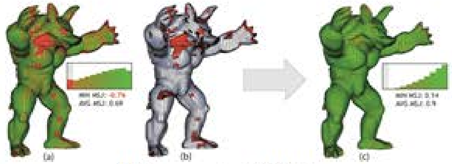
[Knupp 2003]



[Zhang, et al. 2009]

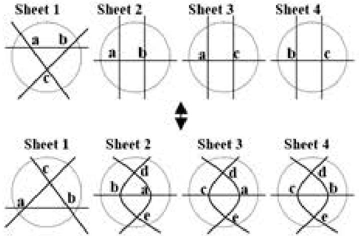


[Ruiz-Gironés, et al. 2014]

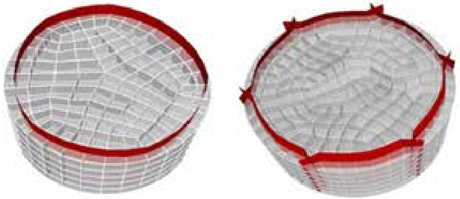


[Livesu, et al. 2015]

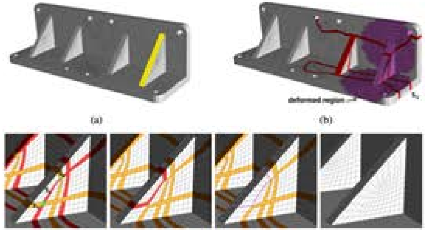
Simplification



[Tautges, et al. 2003]



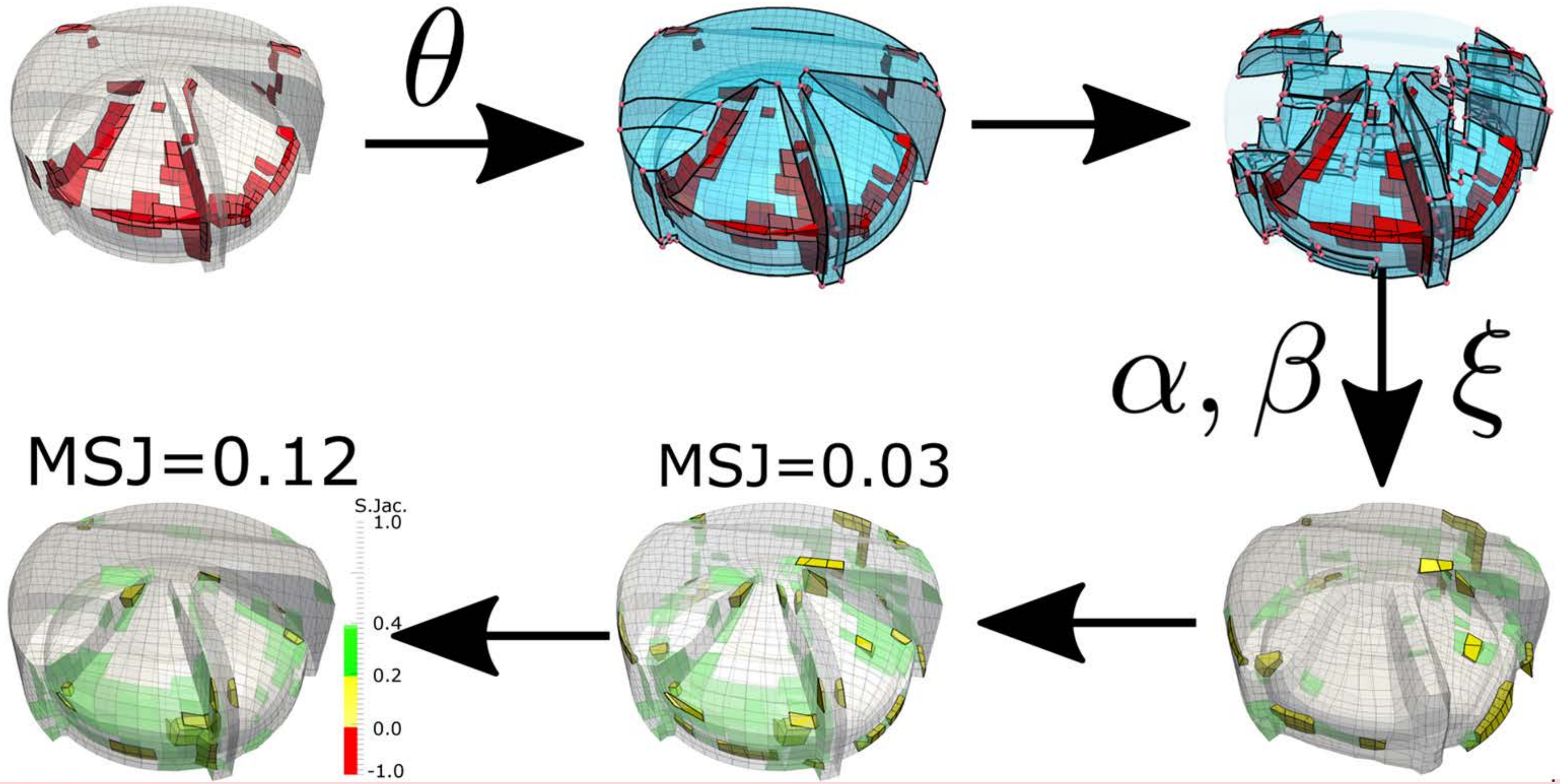
[Ledoux, et al. 2010]



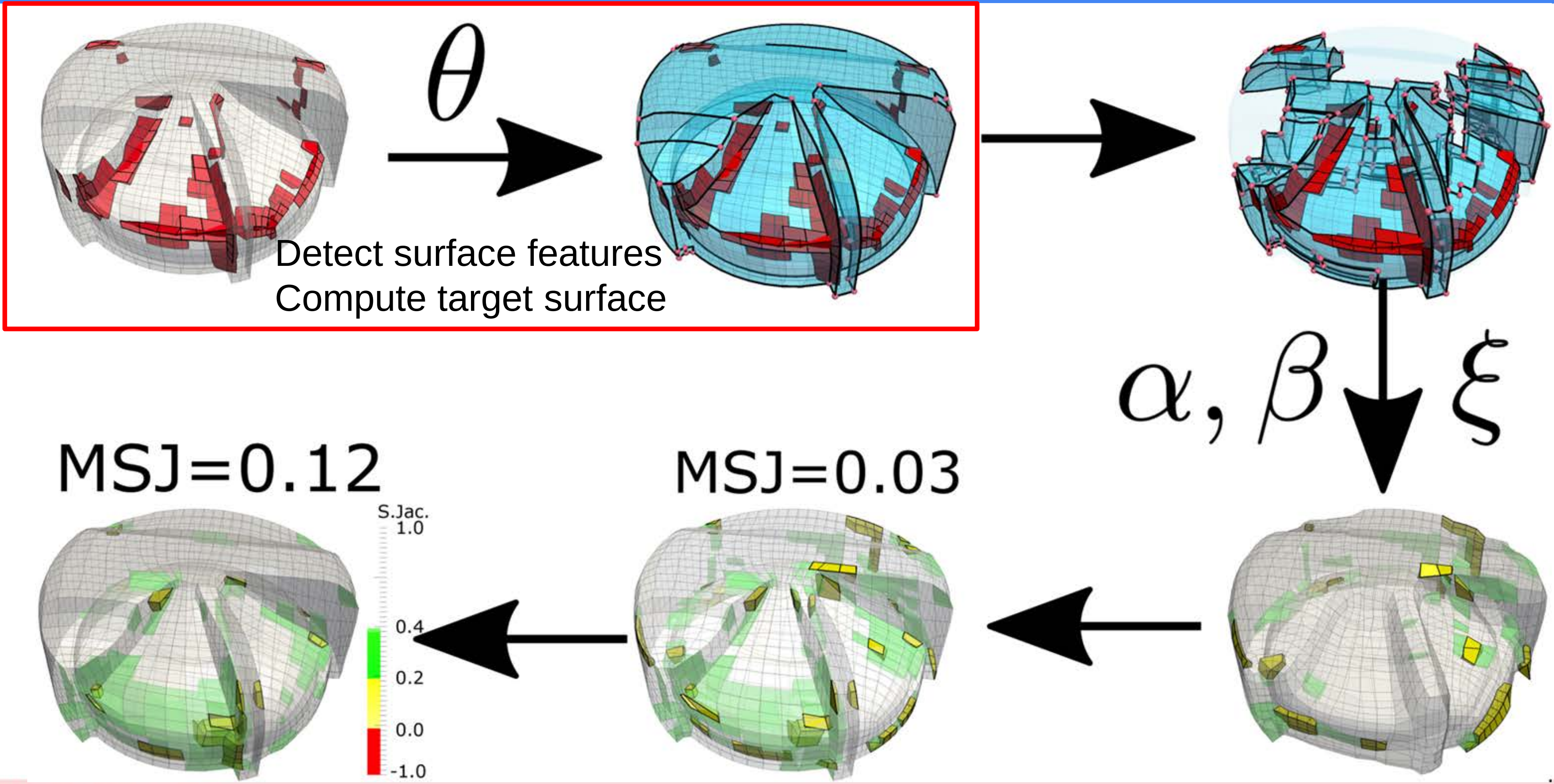
[Zhu, et al. 2014]

Our Method

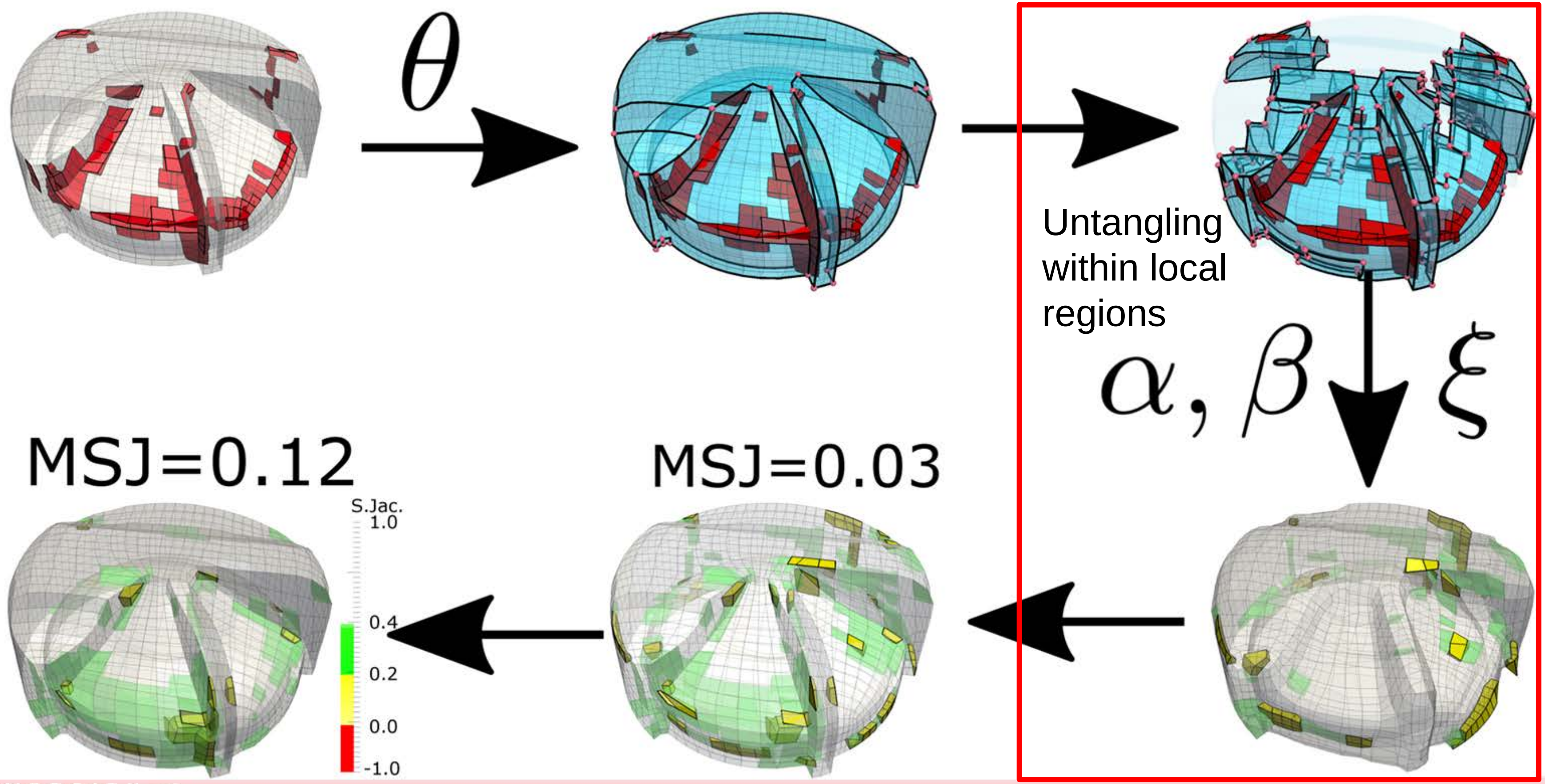
Our Pipeline



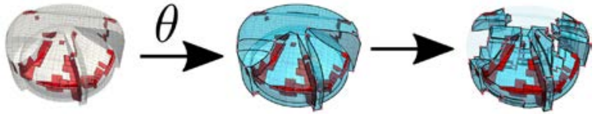
Our Pipeline

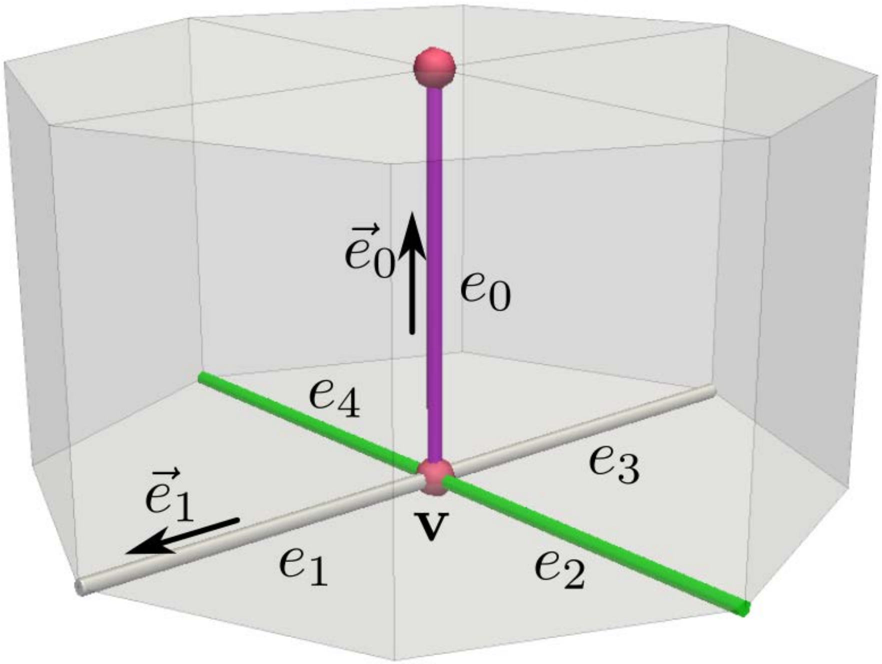


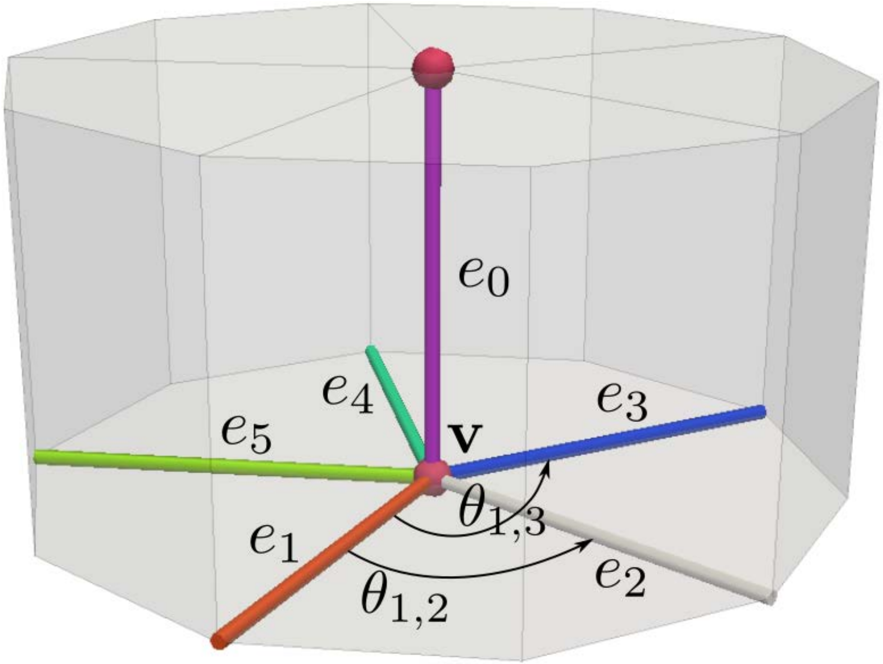
Our Pipeline



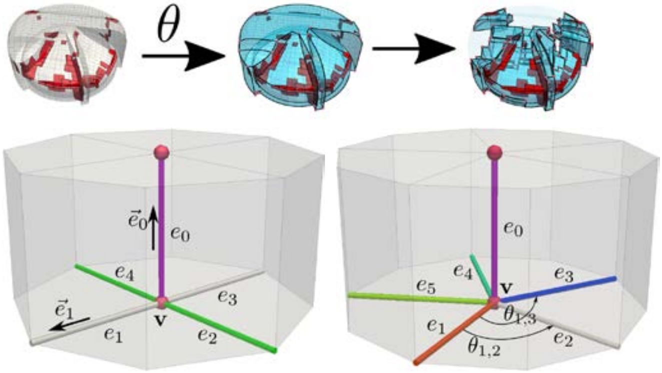
Untangling – Desired angles between edges





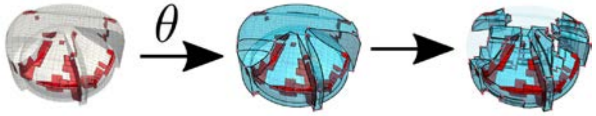


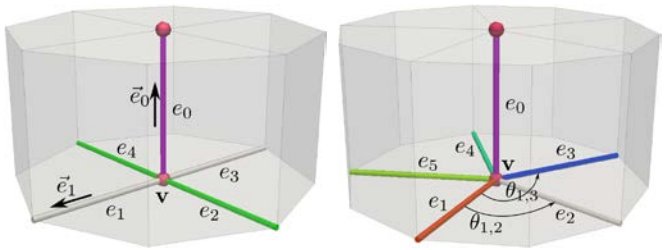
Untangling → unified energy



$$\tilde{E}(\mathbf{v}) = \sum_{e_i \in \mathbf{E}} \sum_{e_i \cap e_j = \mathbf{v}} \left(< \frac{\vec{e}_i}{\|\vec{e}_i\|}, \frac{\vec{e}_j}{\|\vec{e}_j\|} > -\hat{a}\right)^2$$

Untangling → boundary energy





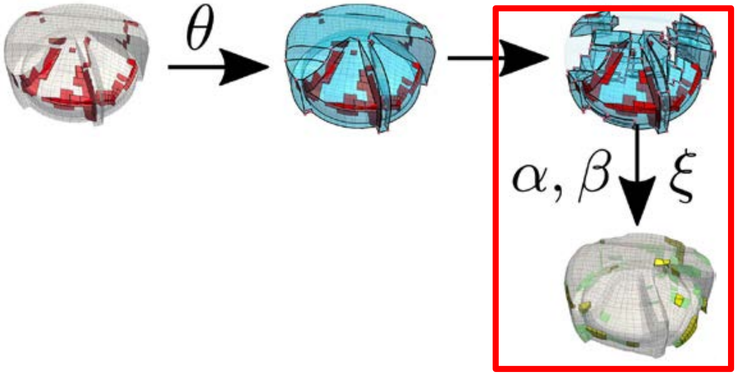
$$
\begin{aligned}
E_{\mathbf{B}}(\mathbf{v}) &= \sum_{v\in S} \beta \|\vec{n} \cdot (\mathbf{v} - \bar{\mathbf{v}})\|^2 \\
&+ \sum_{v\in L} (\alpha \|\mathbf{v} - \bar{\mathbf{v}} - a\vec{t}\|^2 + a^2) \\
&+ \sum_{v\in C} \alpha \|\mathbf{v} - \bar{\mathbf{v}}\|^2
\end{aligned}
$$

See [Livesu et al., 2015] for more details!

Untangling $\rightarrow$ minimize the combined energy

$$\min_{\mathbf{v}} \mathcal{E}(\mathbf{v}) = E_{\mathbf{B}}(\mathbf{v}) + \tilde{E}(\mathbf{v})$$

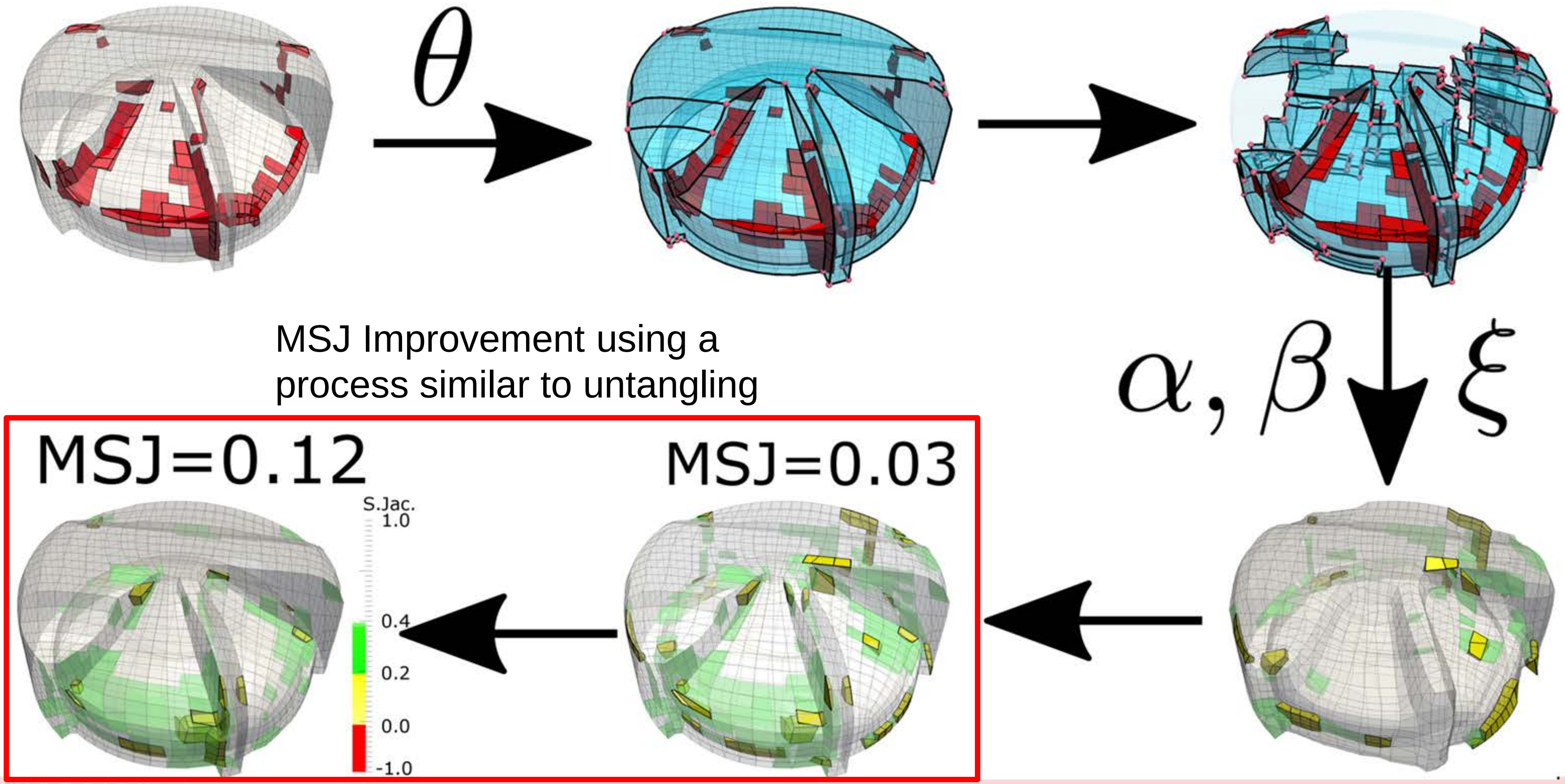
Untangling → untangling algorithm



Algorithm 1: Local untangle

Input: $\mathcal{H}, \Omega_t$
Output: $\mathcal{H}'$
Scale $\mathcal{H}$ and Ω_t ;
Set $\alpha = 1000, \beta = 1000, \xi = 0.6$;
while *current MSJ* ≤ 0 **do**
  while *not reach maximum global iteration (default 20)* **do**
    Identify inverted elements $\mathcal{I}$;
    Extract local regions $\mathcal{R}$ (a copy from $\mathcal{H}$);
    Classify surface vertices for $\mathcal{R}$;
    Compute target edge length by solving Eq. (8) for $\mathcal{R}$;
    $\tau = 1$;
    while *not reach maximum local iteration (default 20)* **do**
      $\tau = 0.9\tau$;
      Solve Eq. (7) for $\mathcal{R}$;
      Save $\mathcal{R}$;
      $\mathcal{R} \leftarrow$ Update vertices. using Eq (9);
      Project surface vertices of $\mathcal{R}$ to its original surface;
      if *#invertedElements increased* **then**
        Recover the saved $\mathcal{R}$;
    end
    Update the vertices of $\mathcal{R}$;
    if *current MSJ* > 0 **then**
      output $\mathcal{H}'$;
  end
  $\xi = \xi - 0.1$;
  if $\xi < 0.2$ **then**
    $\alpha = 0.5 \times \alpha, \beta = 0.5 \times \beta, \xi = 0.6$;
end

Our Pipeline

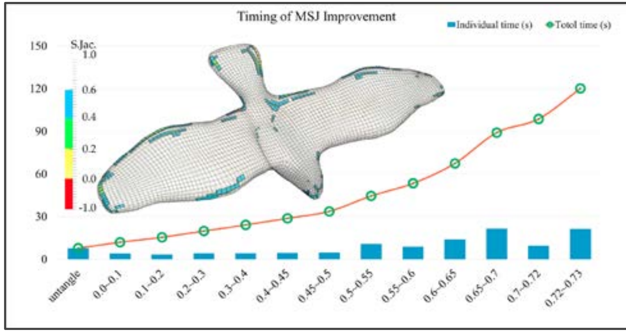


MSJ Improvement using a process similar to untangling

Results

Sampled Results

Performance



Results → Comparison with the edge cone technique

Model	#hexes	#flip	Error	MSJ	ASJ
Armadillo†	29935	323	0.011262	0.14	0.9
Armadillo*	29935	323	0.019349	**0.163**	0.834
Block†	2520	31	0.004555	0.250	0.870
Block*	2520	31	**0.002737**	**0.252**	0.857
Bunny†	37734	1	0.007955	0.606	0.972
Bunny*	37734	1	**0.006477**	**0.651**	0.953
Bust†	5258	30	0.007404	0.114	0.922
Bust(Fig1)*	5258	30	0.008843	**0.201**	0.869
Bust*	5258	30	**0.006795**	**0.183**	0.865
Cap†	4420	50	0.009933	0.106	0.870
Cap*	4420	50	**0.005320**	**0.114**	0.775
Dancing†	35293	5	0.008480	0.354	0.942
Children*	35293	5	**0.006905**	**0.582**	0.931
Hanger†	4539	3930	0.002709	0.716	0.987
Hanger*	4539	3930	**0.001486**	**0.723**	0.974
Impeller†	11174	8857	0.000935	0.184	0.942
Impeller*	11174	8857	**0.000493**	**0.192**	0.934
KingKong†	159488	11	0.010483	0.268	0.967
KingKong*	159488	11	0.016948	**0.500**	0.954

Achieve better MSJ for all cases and better Hausdorff distance error in most cases! ☺

But have lower ASJ ☹

Results → Comparison with AMIPS and edge cone

Table 2. Comparison with AMIPS and edge cone. The first row of each model shows the result by AMIPS † denotes results of edge-cone * denotes ours.

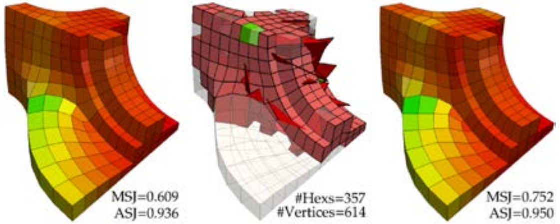
Model	#hexes	input MSJ	MSJ	ASJ
Fertility	10600	0.209	0.46	0.937
Fertility†	10600	0.209	0.478	0.951
Fertility*	10600	0.209	**0.602**	0.933
RockerArm	19870	0.196	0.550	0.923
RockerArm†	19870	0.196	0.556	0.937
RockerArm*	19870	0.196	**0.700**	**0.939**

We achieve much better MSJ for both meshes!

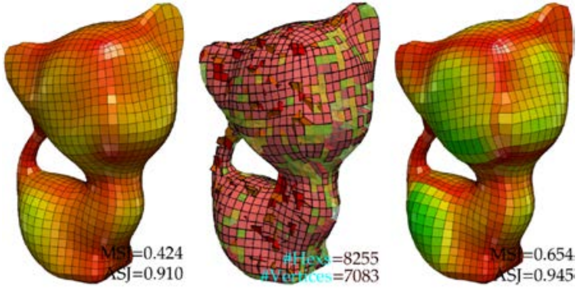
Results → Stress Test

Table 3. Stress Test. Fandisk is created by the frame field method, Kitty is generated by the $\ell_1 - PolyCube$ and airplane is obtained using MeshGems . #flip shows the number of inverted elements after the artificial perturbation. * denotes our results. We use metro tools to compute Hausdorff distance w.r.t. bounding box diagonal '–' means the mesh has no reference surface to compute the Hausdorff distance error.

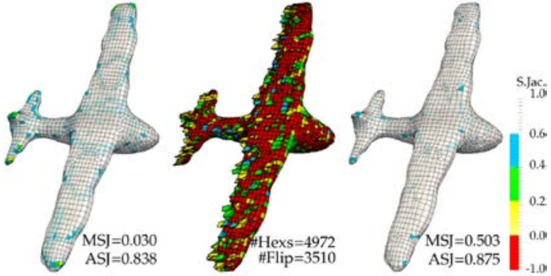
Model	#hexes	#flip	Error	MSJ	ASJ
Fandisk	357	0	–	0.609	0.936
Fandisk*	357	286	0.004769	**0.752**	**0.950**
Kitty	7083	0	–	0.424	0.910
Kitty*	7083	3232	0.012656	**0.652**	**0.937**
airplane1	4972	0	–	0.030	0.838
airplane1*	4972	3510	0.004369	**0.503**	**0.875**



(a) origin mesh (b) stress test input (c) stress test output



(a) origin mesh (b) stress test input (c) stress test output



(a) origin mesh (b) stress test input (c) stress test output

Results → Challenging Cases – Polycube/Octree-Meshes

Table 4. The results on a set of meshes produced from the polycube map database . † denotes results of edge-cone * denotes ours. We use metro tools to compute Hausdorff distance wrt. bounding box diagonal [47]. '–' means the mesh's surface is not in the same scale with the input for computing the Hausdorff distance error.

Model	#hexes	#flip	Error	MSJ	ASJ
airplane1*	17913	467	0.006257	0.731	0.959
bird*	16934	288	0.005774	0.732	0.961
cup1*	16862	40	0.006944	0.723	0.960
chair1*	20344	709	0.004720	0.690	0.941
horse*	44145	304	0.017018	0.600	0.944
blade*	14792	141	0.008885	0.650	0.946
kiss*	19976	247	0.014755	0.500	0.913
bottle1†	15478	127	0.008066	0.132	0.925
bottle1*	15478	127	0.009675	**0.604**	**0.948**
elephant†	46525	421	–	0.012	0.881
elephant*	46525	421	0.009899	**0.500**	**0.915**

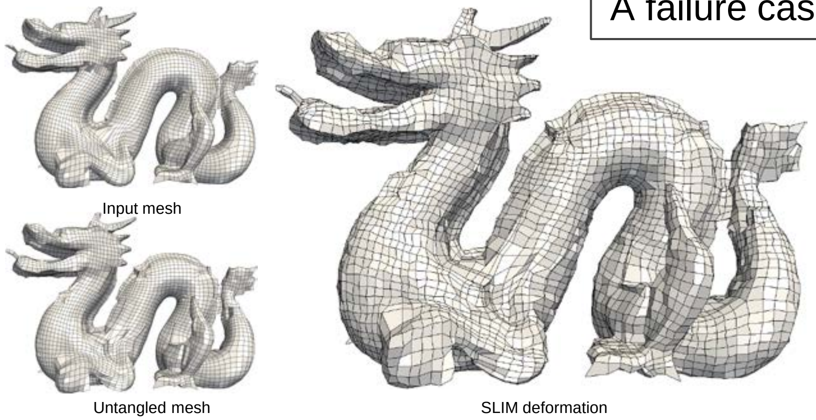
Open source code is available to download !!!

http://www2.cs.uh.edu/~cotrikxu/research/papers/cag2017_hexopt/supplementary_material.zip

Limitations

1. Our method may not improve the average scaled Jacobian substantially.
2. ξ is not the best way to relax the length of edge.
3. surface feature detection is sensitive to the user-specified angle threshold θ

A failure case



Acknowledgment

- Thank the authors of RSF and polycube methods for providing the meshes.
- Thank Marco Livesu for helping generating the edge-cone results for comparison
- Thank all the anonymous reviewers for their valuable comments and suggestions.
- This work was supported in part by NSF IIS 1553329 .

Thank you!