FEATURE-ALIGNED POLY-SQUARE MAPPING OF LARGE-SCALE 2D GEOMETRIES FOR SEMI-STRUCTURED QUAD MESH GENERATION

Celong Liu[1,2] Kelin Hu[3] Qin Chen[4] Xin Li[2*]

[1] *OPPO US Research Center, Palo Alto, CA, U.S.A.*
[2] *School of Electrical Engineering and Computer Science, Louisiana State University, U.S.A.*
[3] *Tulane University, U.S.A.*
[4] *Department of Civil Engineering, Northeastern University, U.S.A.*
[*] *Corresponding Author. Email: xinli@cct.lsu.edu.*

ABSTRACT

We develop a distributed algorithm to compute feature-aligned poly-square maps for large-scale 2D geometric regions and use it to construct low-distortion semi-structured quad meshes. Our proposed algorithm has two main components. The first is a feature-aware graph partitioning that considers workload balancing, minimal communication, geometric regularity, and feature-preserving. The second is a feature-preserved poly-square parameterization. We demonstrate that our algorithm is effective on meshing huge complex coastal/terrain data and can consequently benefit scientific simulations that run on such meshes using high-performance computer clusters.

Keywords: Large-scale Geometric Data Processing, Feature-aware Graph Partitioning, Semi-structured Quad Mesh Generation

1. INTRODUCTION

Generating high-quality mesh is essential to numerical simulation tasks. While unstructured triangular meshes are commonly used due to their ease of generation and good adaptivity, structured or semi-structured paralleletope meshes (e.g., quad meshes for 2D, hex meshes for 3D) are sometimes preferred due to their better support for efficient finite element analysis in many computation-intensive simulations in structural mechanics, fluid dynamics simulations [1]. Here, a structured mesh is one whose vertices all have constant valences and cells are arranged with regular connectivity; and a *semi-structured* mesh, also called *block-structured* mesh, is one that divides the domain into sub-regions, each of which is tessellated by a structured mesh.

This work developed an automatic semi-structured quad meshing algorithm for large-scale 2D planar geometric regions. To process such large geometric data, we develop a *parallel* algorithm to overcome the scale and performance issue. Parallel meshing strategies in existing literature are mainly on the generation of irregular simplex meshes, using either Delaunay-based methods or advancing front techniques to generate triangular or quad meshes [2–4]. Feature preserving is another important issue to consider in many scientific computing tasks. To enforce accurate evaluation or constraints on specific important points, curves, or regions, one would like to have an accurate sampling of these features. However, feature alignment is not always trivial during mesh generation.

For very big or complex geometric region, a divide-and-conquer approach is a natural strategy for parallel processing. We follow the approach similar to [5]. First, we partition the data into solvable sub-regions, considering a trade-off among their size, ge-

ometry, and feature sampling. Then, we map subregions onto canonical poly-square domains for mesh generation. Unlike [5] that constructs meshes locally using advancing front, which results in an unstructured mesh with potential artifacts along partitioning boundary and without supporting feature alignment, our new algorithm now (1) solves the local meshing through a mapping-based method and hence yields a locally structured (globally semi-structured) mesh, (2) integrates feature alignment in partitioning and mapping (mesh) computation, and (3) solves composition through a multi-pass global optimization and hence produces the smooth transition across partitioning boundary and allows flexible insertion of interior singularities to reduce mesh distortion.

The *main contributions* of this paper are as follows. (1) A feature-preserving large-scale semi-structured quad meshing algorithm is developed. Unlike existing large-scale quad meshing algorithms that often cannot handle feature preserving very well, our algorithm can resolve user-determined features effectively. (2) A new feature-aware graph partitioning model is proposed. It can effectively decompose complex geometric models into subregions by considering not just load balancing, minimal communication, but also geometric and feature constraints. (3) Meshes generated using our algorithm could lead to more accurate finite element simulations. Hence, this algorithm can benefit large-scale scientific computing.

2. RELATED WORK

2.1 Quadrilateral Mesh Generation

Quadrilateral mesh generation algorithms can be generally classified into three types, based on the mesh regularity, namely, *unstructured*, *structured*, and *semi-structured* meshes.

Structured Quad Meshing. Rigorously speaking, a fully-structured quad mesh is a tensor-product patch that deforms from an $N \times M$ grid. However, such a fully regular grid topology is often too restrictive: wrapping such a grid onto a region with a complex topology is infeasible and covering a shape with long and thin branches will result in severe distortion. Therefore it is essential to introduce some singularities or unstructured organization to a more flexible balance the trade-off between regularity and distortion.

Unstructured Quad Meshing. Commonly used and extensively studied quad meshing algorithms are on generating unstructured meshes. Classic algorithms include those based on triangle-to-quad conversion [6, 7], quad-tree projection [8, 9]. A widely studied strategy is the advancing front algorithm that starts the construction of cells from the boundary, then propagates layer inwards by layer, until the propagation fronts quench each other near media axes and form singularity curves. Notable advancing front algorithms include the classic paving [10], Q-Morph [11], H-Morph [12], to the recent frontal Delauney algorithm [13]. Another group of effective algorithms developed recently in the graphics field in constructing low-distortion surface parameterization and meshing is based on building a smooth cross-frame (2-rotational symmetry) field. This strategy has been very successfully in building low-distortion unstructured quad meshes, as demonstrated by multiple recent algorithms, such as QuadCover [14], Mixed-integer quadrangulation [15], Integer-grid maps [16], singularity-restricted field [17], QMorph Cross Field [18]. These methods can produce meshes with very desirable (i.e., low-distortion) element shape quality. However, the unstructured nature of these meshes sometimes still hampers their performance (compared with using more structured meshes) in scientific computing.

Semi-structured Quadrilateral Mesh Generation. Semi-structured or block-structured meshes divide the domain into sub-regions, call blocks, each of which is tessellated by a structured grid. The global layout of these blocks is often unstructured and can be encoded using a graph with irregular connectivity. The semi-structured mesh can be constructed on an arbitrary complex domain and offers a flexible trade-off between regularity and element distortion. The macro-unstructured-micro-structured hierarchy layout of a semi-structured mesh also naturally supports parallelization where the calculations on local blocks can be executed on parallel processors fully utilizing the numerical efficiencies of its regularity. On curved surfaces, recent frame-field approaches [19,20] use rectangle patches to generate high quality semi-structured meshes, however they are hard to be extended to large scale geometric data. Polycube parameterizations have been used to generate semi-structured meshes where each rectangular sub-patch is tessellated regularly, and the global layout is described by the topology of the polycube polyhedron. Multiple semi-automatic or automatic polycube parameterization algorithms have been developed for quad- or hex-meshing [21–26]. However, with polycube parameterization, the singularities locate at the polycube corners. Distortion is often significant near these singularities. Furthermore, solving optimal polycube parameterization is usually expensive and prohibitive to be directly applied to large-scale geometric data.

2.2 Quad Meshing by Divide-and-Conquer

To process large-scale geometric data, we adopt a domain decomposition approach that partitions the data into solvable subparts for distributed and parallel processing. Related to our approaches are var-

ious divide-and-conquer algorithms in mesh generation. For example, medial axis decomposition was used to partition geometric regions [27, 28]. The general ideas of these approaches are first to extract medial axis of regions then insert cuts on them to cut the region into smaller "shape atoms" which could be meshed using templates. Voronoi diagrams have also been used [5, 29] to generate coarse cells upon which quad meshes can be constructed. On curved surfaces, quad patch layout design has been studied to facilitate the construction of low-distortion quad meshing of surfaces. The Morse-Smale Complex has been used [30, 31] to design quadrilateral patches. More generally, the cross-frame field on the surface can be optimized to generate principal-direction-aligned quad layouts [19]. These geometry-adaptive quad patch layouts have been shown effective in generating high-quality quad meshes for curved surfaces. The downside of these algorithms is their computational complexity. When dealing with large-scale geometric data, globally solving a non-linear (mixed) integer optimization would be infeasible.

2.3 Feature Alignment Quad Meshing

In some physical simulation problems, we often need to preserve the feature regions in the generated mesh for special study. T-mesh is a widely used approach to generate feature aligned quadrilateral mesh [32–34]. But T-junction is not acceptable in some largely scaled physics simulation system [35]. Another approach to generating feature aligned mesh is cross-field based parameterization [36, 37]. They can generate high-quality feature aligned quadrilateral mesh. However, the unstructured nature and their algorithm are not easy to be parallelized for largely scaled mesh generation. In this paper, we design a distributed and parallel meshing framework that can effectively generate high-quality feature aligned/preserved quadrilateral mesh.

3. FEATURE-AWARE GRAPH PARTITIONING

To effectively process a very big and complex geometric region, we first do a data partitioning to split the data into small, solvable subparts. In [38], a graph partitioning is solved to obtain subregions for quad mesh generation. An algorithm was proposed to optimize three criteria: (1) workload balancing where subregions should have similar sizes; (2) small communication cost where length of subregion boundary should be minimized; and (3) geometry regularity where each subregion should have corner angles close to $k\pi/2$, $k \in \{0, 1, 2, 3\}$. In our partitioning task, all these three criteria are still desirable, and we further consider one more criterion on feature preservation. To have features points or feature curves sampled in the final mesh, we can partition the geometry so that subregion boundaries go through these features since all the boundary elements will be exactly sampled in the final mesh. So we have the four criteria formulated as (4) the partitioning separators should go through feature curves as much as possible.

3.1 Notation and Recap of Geometry-aware Partitioning

We adopt the notations from [38] and recap that partitioning algorithm first before we explain our modification to the algorithm in the next section.

Given a triangulated 2D region $M = (V^M, E^M, F^M)$, where V^M, E^M, F^M are the sets of vertices, edges, and faces (cells), respectively, let $G = (V^G, E^G)$ denote its dual graph, where V^G, E^G are the sets of nodes and arcs. The weight of a node $v \in V^G$ is defined to be the area of its associated cell $f \in F^M$, and the weight of an arc is the length of its associated edge. A k-way partitioning on M can be computed on G. Without losing generality, we explain its idea using the 2-way partitioning in the following (which can be directly generalized to k-way [5]) that partitions G into two sub-graphs $G_0 = (V^{G_0}, E^{G_0})$ and $G_1 = (V^{G_1}, E^{G_1})$, where $V^{G_1} = V^G \setminus V^{G_0}$. An indicator variable x_i is assigned to each node $v_i^G \in V^G$,

$$x_i = \begin{cases} 0, & \text{if } v_i^G \in V^{G_0} \\ 1, & \text{if } v_i^G \in V^{G_1} \end{cases}.$$

Then for each arc $e_{ij}^G = [v_i^G, v_j^G]$, we assign a variable $y_{ij} = x_i - x_j$:

$$y_{ij} = \begin{cases} 0, & \text{if } v_i^G \text{ and } v_j^G \text{ are in the same sub-graph} \\ 1 \text{ or } -1, & \text{otherwise} \end{cases}.$$

We have $\mathbf{y} = \mathbf{U}\mathbf{x}$, where $\mathbf{x}$ and $\mathbf{y}$ are node and arc variable vector respectively, and $\mathbf{U}$ is an $|E^G| \times |V^G|$ matrix.

Firstly, to **balance the workload** and avoid big area difference between subregions, we have the following constraint

$$c_1 \leq \mathbf{x}^T \mathbf{w}_v - c \leq c_2, \tag{1}$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ is the variable vector, $\mathbf{w}_v = (w_{v_1}, w_{v_2}, \ldots, w_{v_n})^T$ is the node weight vector, $c = \frac{1}{2}\sum_i w_{v_i}$, and c_1, c_2 are the constant thresholds. In our experiments, we set $c_1 = c_2 = 0.1c$.

Secondly, a smaller total separator length means less inter-process communication in parallel computing. Therefore, it is desirable to minimize the **total separator length**

$$L_S = \mathbf{y}^T \mathbf{W}_e \mathbf{y} = \mathbf{x}^T \mathbf{U}^T \mathbf{W}_e \mathbf{U}\mathbf{x}, \tag{2}$$

where $\mathbf{y} = (y_{e_1}, y_{e_2}, \ldots, y_{e_n})^T$ is the edge variable vector, $\mathbf{W}_e = \text{diag}(w_{e_1}, w_{e_n}, \ldots, w_{e_n})$ is a diagonal matrix composed of arc weights. Combining these above criteria one and two we have the common graph partitioning formulated on triangular meshes.

Thirdly, in quad meshing, it is desirable to have angles between separators close to $\frac{k\pi}{2}$. Ideally, we can use a *separator angle term* to penalize each such angle's deviation from $\frac{k\pi}{2}$. Consider two edges $e_i, e_j \in E^M$ of the original mesh M, we use $Inc(i,j) = 1$ to denote e_i and e_j are incident, and $Inc(i,j) = 0$ if they are not. If $Inc(i,j) = 1$, suppose they form an angle $\theta_{i,j}$. We define an *angle deviation function*

$$\delta_{\theta_{i,j}} = \begin{cases} \min_k\{|\theta_{i,j} - \frac{k\pi}{2}|, k \in \{0,1,2,3\}\}, & \text{if } Inc(i,j) = 1 \\ 0, & \text{if } Inc(i,j) = 0 \end{cases} \tag{3}$$

to describe the deviation from angle $\theta_{i,j}$ to the nearest $\frac{k\pi}{2}$ angle. The *accumulated separator angle deviation* can then be formulated as

$$D_\theta = \mathbf{y}^T \mathbf{W}_\theta \mathbf{y} = \mathbf{x}^T \mathbf{U}^T \mathbf{W}_\theta \mathbf{U} \mathbf{x}, \tag{4}$$

where $\mathbf{y} = (y_{e_1}, y_{e_2}, \ldots, y_{e_n})^T$ is the edge variable vector, and

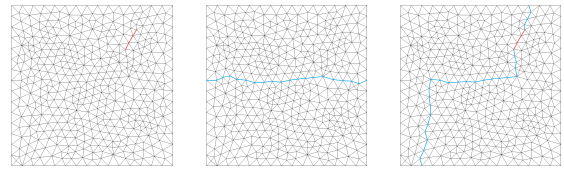
$$\mathbf{W}_\theta = \begin{pmatrix} 0 & \delta_{\theta_{1,2}} & \delta_{\theta_{1,3}} & \ldots & \delta_{\theta_{1,n}} \\ \delta_{\theta_{2,1}} & 0 & \ldots & \ldots & \delta_{\theta_{2,n}} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \delta_{\theta_{n,1}} & \ldots & \ldots & \ldots & 0 \end{pmatrix}$$

is an $|E^G| \times |E^G|$ matrix storing deviation angles $\delta_{\theta_{i,j}}$.

However, this dense angle deviation matrix $\mathbf{W}_\theta$ need to be computed from all the edge pairs of the tessellation M, and directly minimizing this angle deviation term is very expensive. In [38], a two-stage approximate algorithm is adopted. They first tessellate the 2D geometry using L_∞-CVT whose cells have near-90 deg angles, then they perform a common graph partitioning on these cells to get an approximate partitioning.

3.2 Feature Curve Preservation

To preserve feature curves/points in the final mesh, one approach is to let the partitioning separators go through the features because the sampling of these separators (subregion boundaries) can directly and easily guarantee as boundary constraints in the subsequent parameterization step. In Fig. 1(a), a feature curve is indicated inside the mesh as red color and we want the final partitioning separator go through this curve. In Fig. 1(b), the separator (in blue color) roughly divides the region from the middle and does not pass the curve since the partitioning does not have a feature curve preservation mechanism. In Fig. 1(c), by applying the feature preservation term we will introduce in Eqn. 6, we can get the resultant separator pass the feature curve.



(a) (b) (c)

Figure 1: Partitioning results with or without feature preserving. (a) A rectangle region with feature lines highlighted, (b) partitioning without features preserved, (c) partitioning with features preserved.

Therefore, we incorporate a **feature-preserving term** into the graph partitioning algorithm as a soft constraint, to attract the partitioning boundary towards features as much as possible. As a soft constraint, some features may not be on subregions boundaries, and they will be handled in the next step during mapping computation.

A feature curve is discretely represented as a list of edges $\{e_{f_1}, e_{f_2}, \ldots, e_{f_m}\} \in E^M$, where m is the total number of feature edges, f_i is the index of the edge in E^M. We first define a feature edge indicator

$$\sigma_{e_i} = \begin{cases} 1, & e_i \in \{e_{f_1}, e_{f_2}, ..., e_{f_m}\} \\ 0, & otherwise \end{cases}, \tag{5}$$

then, define the feature-preserving term as

$$D_f = \sum_{i=1}^{m} y_{e_{f_i}}^2 = \mathbf{y}^T \mathbf{W}_f \mathbf{y} = \mathbf{x}^T \mathbf{U}^T \mathbf{W}_f \mathbf{U} \mathbf{x}, \tag{6}$$

where $\mathbf{y} = (y_{e_1}, y_{e_2}, \ldots, y_{e_n})^T$ is the edge variable vector, and $\mathbf{W}_f = \text{diag}(\sigma_{e_1}, \sigma_{e_n}, \ldots, \sigma_{e_n})$ is an $|E^G| \times |E^G|$ diagonal matrix. To get $\mathbf{W}_f$, we just need to traverse all the edges once.

If a feature line e_i locates on a sub-region's boundary, then its contribution to D_f is 1 since its edge variable $y_{e_i} \neq 0$ and $\sigma_{e_i} = 1$. On the other hand, for a non-feature edge or an edge that is not on the boundary, either its y_{e_i} or σ_{e_i} is zero, so its contribution to D_f is 0.

Thus, to make the separators align with most of the feature edges, we need the edge variables $\mathbf{y}$ which can minimize $-D_f$.

3.3 Formulation of the Optimization

Finally, the partitioning reduces to minimizing the separator angle deviation and feature-preserving penalty together with the total separator length, subject to the workload balance constraint, namely,

$$\min_{\mathbf{x} \in \{0,1\}^n} E(\mathbf{x}) = L_S + \lambda_1 D_\theta - \lambda_2 D_f, \text{subject to Eq. (1)}, \tag{7}$$

where λ_1, λ_2 are weighting factors. In our experiments, we simply set $\lambda_1 = 1, \lambda_2 = 1$.

This is an NP-hard integer programming problem. Solving such an optimization problem accurately is prohibitively expensive for large graphs. So in scientific computing, where meshes often contain millions or billions of vertices, a more efficient approximate solving algorithm is needed.

3.4 Approximate algorithm

METIS [39] is a widely used open-sourced program for partitioning unstructured graphs. METIS using is a heuristic algorithm called Hill-Scanning, which can iteratively move vertices to corresponding groups and keep the balance of the partitioning by checking the energy gaining.

To have the feature-preserving term minimized, we modified the algorithm of METIS and developed a new solver incorporating the feature terms. We adopt the Metropolis-Hastings algorithm [40] which are widely used in stochastic optimization problems such as Monte-Carlo simulation, etc. For each possible moving of the vertices, we evaluate the change of energy, ΔE, (Eqn. 7). If $\Delta E < 0$, then we will accept this moving. Otherwise, if $\Delta E > 0$, we will perform a random test:

- Generate a random number $p \in [0, 1]$
- Test whether $p < e^{-\lambda_{MH}\Delta E}$, if so, accept the moving; otherwise reject it.

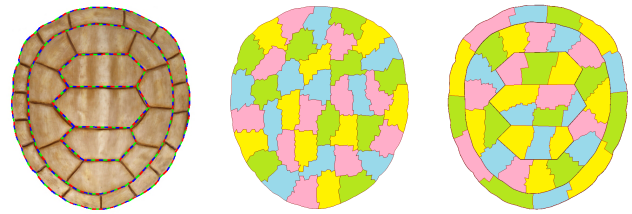
In our experiments, we set the λ_{MH} to a small number 0.01, so we can accept more energy increase move to avoid local minima.

The advantage of this stochastic optimization strategy is that it can be easily parallelized and give us very good partitioning results.

Fig. 2 shows an example of partitioning that incorporates feature curves. (a) shows a turtle shell region with feature curve highlighted. Without considering these features, the partitioning result is shown in (b). The feature-preserved partitioning result is shown in (c).

4. FEATURE-PRESERVED POLY-SQUARE MAPS

After partitioning, we can construct quad meshes on sub-regions in parallel. This section illustrates our local mesh construction. To ensure the element consistency across the subregion boundaries, we would select meshing algorithms that allow us to enforce a pre-determined boundary constraint, rather than using free-boundary parameterization algorithms, which is difficult to control and refine. One straightforward meshing strategy is to conduct an advancing front algorithm [10] to generate meshes on subregions from their consistently pre-sampled subregion boundaries. Despite its simplicity and efficiency, the advancing front algorithm has several *limitations*: (1) it only generates unstructured meshes; (2) the element quality near the media axes is hard to control and saliently worse than those near the subregion boundary; (3) for elements near subregion boundaries, their directional smoothness across the boundary could be undesirable, due to that the meshes are generated individually and in a one-pass manner. To overcome these three limitations to generate better-structured meshes and obtain a better control on mesh quality and boundary smoothness, we adopt a mapping based meshing strategy by computing a poly-square parameterization.



(a) (b) (c)

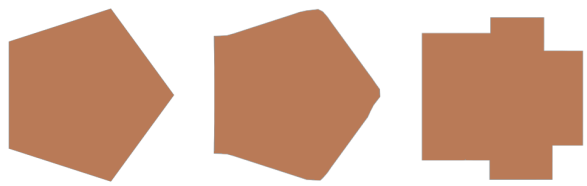
Figure 2: Partitioning results with or without feature preserving. (a) A turtle shell region with feature lines highlighted, (b) partitioning without features preserved, (c) partitioning with features preserved.

4.1 Poly-square Parameterization

A *poly-square* is a 2D orthogonal polygon whose edges are all parallel to coordinate axes. The 3D analogy of the poly-square parameterization is polycube parameterization, which has been extensively studied in graphics and computer-aided design literature [23–26, 41] and applied in texturing spline construction, and mesh generation (for 3D surfaces). Our partitioning produces 2D subregion with relatively simple geometry. So we follow the polycube parameterization algorithm of [41] and implemented it in 2D.

Each local subregion M is described by its boundary ∂M. First, we tessellate it using a triangle mesh $\bar{M} = \{T, \bar{X}\}$ conforming to this boundary, where $T = \{t_i\}$ and $\bar{X} = \{\bar{x}_i\}$ are the sets of triangles and vertex coordinates, respectively. The output is a deformed triangle mesh $N = \{T, X\}$ where X is the new corresponding vertex positions. ∂N is a poly-square. This deformation is solved by minimizing an objective function consisting of two terms: (1) boundary normal alignment error, and (2) element distortion.

Boundary Normal Alignment. The boundary vertices should be deformed to align with coordinate axes to make a poly-square. In Fig. 3(a), we have a pentagon region to be deformed. Only when we align its boundary normal to coordinate axes, the result will be a valid poly-square (Fig. 3(b)). Otherwise (Fig. 3(c)) it cannot become a poly-square.



(a) (b) (c)

Figure 3: Boundary Normal Alignment for Poly-Square. (a) A pentagon before deformation; (b) deformation without boundary normal alignment; (c) deformation with boundary normal alignment.

This can be formulated by minimizing the L_1-norm deviation,

$$d(\mathbf{n}) = \|\mathbf{n}\|_1 - 1,$$

where $\|\mathbf{n}\|_1 = \sum_i^k |n_i|$ is the L_1-norm of vector $\mathbf{n}$. Given a boundary edge $e \in \partial N$, whose normal is $\mathbf{n}_e$, its deviation from axis coordinates is $d(\mathbf{n}_e)$. The accumulative boundary normal alignment error over the entire ∂N can be written as:

$$E^B(X) = \sum_{e_i \in \partial N} L(e_i)(\|\mathbf{n}_{e_i}\|_1 - 1). \tag{8}$$

where $L(e)$ is the length of e.

Deformation Distortion. To suppress the element's shape distortion during deformation, we shall also minimize the deformation distortion. In [41], the MIPS energy [42] is used to measure the distortion. On a triangle t_i, the MIPS distortion is defined as $\epsilon_{MIPS}(t_i) = \frac{\sigma_{i,1}}{\sigma_{i,2}} + \frac{\sigma_{i,2}}{\sigma_{i,1}}$, where $\sigma_{i,1}$, $\sigma_{i,2}$ are the two singular values of the Jacobian of transformation on triangle t_i. In [5], it is observed that having a heavier penalty on the severely distorted element is desirable to produce better shaped poly-square domain topology. Hence, they used an exponential form of MIPS, $\epsilon_{EMIPS}(t_i) = e^{\frac{\sigma_{i,1}}{\sigma_{i,2}} + \frac{\sigma_{i,2}}{\sigma_{i,1}}}$, to measure the distortion on triangle t_i. In [43], an Advanced MIPS energy (AMIPS) was developed, and it used the exponential terms to penalize the angle distortion as well. It also contains an area distortion term and allows us to find a balance between these two terms. AMIPS is formulated as

$$\epsilon_{AMIPS}(t_i) = e^{\alpha s \cdot \frac{1}{4}(\frac{\sigma_{i,1}}{\sigma_{i,2}} + \frac{\sigma_{i,2}}{\sigma_{i,1}}) + (1-\alpha) s \cdot \frac{1}{2}(det(J(t_i)) + det(J(t_i))^{-1})}$$

where $J(t_i)$ is the Jacobian matrix of transformation on t_i, α is a parameter controlling the weight of angle distortion and area distortion, and s is a parameter controlling the level of penalty. This AMIPS energy allows us to control the area distortion better. A small s has little effect on penalizing the maximal distortion, and a large s will cause numerical instability. Therefore, we adopted AMIPS in this pipeline. In our experiment, we choose $\alpha = 0.5, s = 5$. Accordingly, the accumulative AMIPS distortion on the entire mesh can be computed as $E^D(X) = \sum_{t_i \in T} \epsilon_{AMIPS}(t_i)$.

Finally, combining these two terms, the poly-square deformation reduces to solving the following optimization problem:

$$\phi = \text{argmin}_X \; E^D(X) + \lambda E^B(X), \tag{9}$$

where λ is the weighting factor balancing the two different emphases of the Domain shape optimization. A small λ indicates more emphasis on minimizing distortion and could result in a non-orthogonal and invalid poly-square. A big λ, in contrast, produces a valid poly-square by possibly introducing larger distortion and more singularities.

In all our experiments, we use $\lambda = 5$ which is the balance point that can give a valid poly-square and least singularities.

We solve this problem using the Hybrid L-BFGS (Limited Memory Broyden-Fletcher-Goldfarb-Shanno Method) optimization framework [44].

4.2 Preserving Feature Lines

To ensure feature (lines) to be sampled in the final mesh, we require the feature points to be mapped to integer coordinates. In the data partitioning stage, feature lines are incorporated as a soft constraint. So some feature lines, not sampled as boundary region separators, need to be resolved here during parameterization.

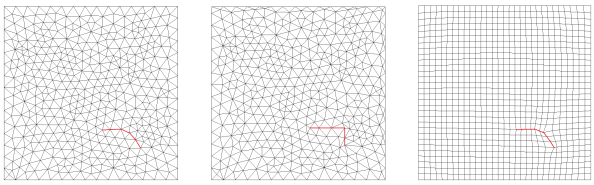
To preserve feature lines in poly-square parameterization, we just need to force the sampled feature points to be deformed to integer coordinates.

To enforce this integer constraints, first, we add a soft constraint, to minimize the deviation of deformed feature lines' directions (or their normals) from axes coordinates. Namely, for feature edges $F = \{e_{f_1}, e_{f_2}, \ldots, e_{f_m}\}$, we have the third penalty term,

$$E^F(M) = \sum_{e_i \in F} L(e_i)(\|\mathbf{n}_{e_i}\|_1 - 1). \tag{10}$$

Then finally, after the optimization, we round the resultant coordinates of each feature line segment's end nodes to integers. Fig. 4 shows an example of feature-preserving in mapping computation. The red curve is the feature to sample. After deformation, they are

mapped onto horizontal or vertical line segments with integer coordinates.



(a) (b) (c)

Figure 4: Poly-square mapping to mesh a square domain with features (red curve). (a) The original triangular mesh, (b) the deformed mesh, and (c) the final quad mesh.

4.3 Initialization and Pre-orientation

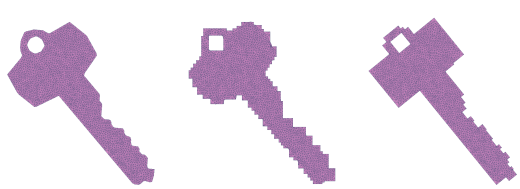
The deformation-based algorithm transforms each boundary edge to its nearest coordinate axis. So this deformation is affected by the shape's initial orientation. Optimizing a pre-orientation of M before the deformation could provide a good initialization for the poly-square parameterization and lead to a better mapping result.

Adopting the strategy of [45], we select the pre-orientation that makes the directions of M's boundary line segments to have the smallest deviations from coordinate directions. More specifically, suppose we use a function Φ to measure the deviation of an edge's normal $\mathbf{n}$ from a coordinate axis, $\Phi(\mathbf{n}) = n_x^2 n_y^2$ where n_x and n_y are $\mathbf{n}$'s x and y components respectively.

Then an optimal pre-orientation, rotation R, minimizes the following function,

$$\hat{R} = \text{argmin}_R \sum_{e_i \in \partial M} \Phi(R \cdot \mathbf{n}_i). \tag{11}$$

The 2-dimensional rotation R can be represented using an angle and the above problem can be solved using gradient descent method efficiently.



(a) (b) (c)

Figure 5: Pre-orientation for Poly-square Mapping. (a) The original mesh; (b) resultant poly-square without pre-orientation; (c) resultant poly-square with pre-orientation.

Fig. 5 shows an example of how pre-orientation improves the poly-square map. The direct poly-square deformation performed on the model (a), without pre-orientation, results in (b). With a pre-orientation, the deformation results in (c) which have fewer boundary corners, and also, smaller mapping distortion.

5. CONQUERING LOCAL MAPS INTO GLOBAL PARAMETERIZATION

5.1 Global Poly-square Map (GPM)

When processing a large or complicate region M, if we directly compute its poly-square parameterization, the optimization becomes too expensive. With the previous data partitioning, one simple strategy is to solve the poly-square parameterization distributedly.

Suppose the given domain M is partitioned into subregions $M = \sum_i \cup M_i$. For each adjacent subregion pair M_u and M_v, their separator is a common boundary curve L_{uv}, which is defined by a set of common vertices $\{X_{uv}\} = X_u \cap X_v$. Since each subregion will be mapped onto a poly-square domain on a separate processor. We need to enforce the following consistency constraint on common separators to keep boundary consistency:

$$x^u_{I^{uv}_{u,i}} = x^v_{I^{uv}_{v,i}}, \forall i = 1, ..., N_{uv}, \tag{12}$$

where $I_{u,i}$ is the index of the i-th vertex from L_{uv} in M_u, $I_{v,i}$ is the index of the i-th vertex from L_{uv} in M_v, N_{uv} is the total number of vertices on L_{uv}. This constraint means each pair of corresponding vertices on the shared subregion boundary should have the same image after their deformations. Although this map is solved in a distributed way, this strict boundary constraint ensures the resultant parameterization is a globally continuous poly-square map. In the following, we denote this computed parameterization a *Global Poly-square Map* (**GPM**).

Fig. 6 demonstrate the GPM updating process for the sub-regions of the key, for the clearly visualization purpose, we use 8 subregions to do this demonstration. We can see that all the regions will continuously deform to a poly-square shape and the shared subregion boundaries will keep the same shape and orientation during all the time.

5.2 Piecewise Poly-square Map (PPM)

The poly-square map has all its singularities on the poly-square's corners. So usually, these regions have big distortion and cells near these singularities may be severely sheared (See Fig. 13 (b,e,h) for examples). To reduce the mapping distortion, designing a mechanism to allow placing interior singularities at suitable places is often very helpful.

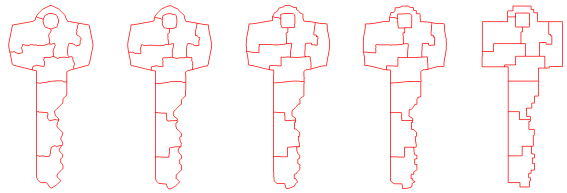


Figure 6: The deformation process of a GPM algorithm. (a) The partitioning before deform. (b) Iteration number = 1. (c) Iteration number = 5. (d) Iteration number = 10. (e) Final poly-square result (Iteration number = 209)

In our computational pipeline, with data partitioning, we get subregions that have near perpendicular corner angles, which is desirable for poly-square parameterization. More importantly, we can build upon this partitioning a mechanism that could support the flexible insertion of interior singularities.

Following the previous notations, instead of using Equation 12, we enforce the following new consistency constraint on common separators:

$$L(I_{u,1}, I_{u,k}) = L(I_{v,1}, I_{v,k}), \forall k = 2, \ldots, N_{uv}, \quad (13)$$

where $I_{u,i}$ is the index of the i-th vertex from L_{uv} in M_u, $I_{v,i}$ is the index of the i-th vertex from L_{uv} in M_v, $L(I_1, I_2)$ is the curve length from point I_1-th vertex to I_2-th vertex, N_{uv} is the total number of vertex on L_{uv}. This constraint means that for each pair of vertices on the common separator, their curve length on both subregions, after deformation, should be the same.

Interior Singularities. We do not require the parametric coordinates from two subregions to exactly overlap, and on the poly-square corners we We do not require the sum of its incident angles to be exactly 2π. For a corner whose surrounding angles do not sum up to 2π, we generate an interior singularity. We call this poly-square parameterization that allows interior singularities a *Piecewise Poly-square Map* (**PPM**).

Parallel Computation. Since different subregions are processed separately, during a subregion's deformation, its neighboring regions' deformation is unknown. If we process all the subregions simultaneously, the consistency constraint cannot be directly enforced. Therefore, we perform deformation in a multi-pass manner. All the non-adjacent subregions can deform simultaneously; then their new boundary geometry is propagated to adjacent subregions, which will be used as initial poses of their deformations in the next round. To do this, we perform a graph coloring algorithm [46] to divide all the subregions $\{S_1, S_2, ..., S_p\}$ into k group $\{G_1, G_2, ..., G_k\}$. Each group consists of non-adjacent subregions and can be processed in parallel simultaneously.

Propagation on Atlas until Convergence. To support a more efficient propagation of boundary condition from a subregion to its neighbors, we construct a *multi-layer buffer-zone* around each subregion. From the boundary of subregion ∂M_i, we grow outwards to construct an offset contour ∂B_i, and call the region bounded by ∂M_i and ∂B_i the *buffer zone* B_i. In Fig. 7, we illustrate the relationship between a subregion M_i and its buffer zone B_i. For simplicity, we can construct B_i using the k-ring elements from ∂M_i. Then we call the enlarged patch $C_i = M_i \bigcup B_i$ the *deformation chart*. During iterative deformation, instead of only deforming M_i, we deform C_i subject to two constraints (1) ∂M_i should deform to an orthogonal polygon, and (2) the vertices on ∂C_i should not move. Namely, we solve

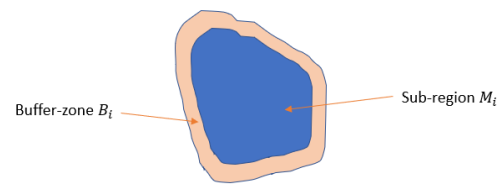


Figure 7: A subregion M_i and its buffer zone B_i.

$$\phi_i = \text{argmin} E^D(C_i) + \lambda E^B(M_i), \text{ s.t. } \partial C_i \text{ being fixed.} \quad (14)$$

In the k-th iteration, M_i deforms to make itself orthogonal while vertices on ∂C_i are fixed to the positions from their last iteration. This ensures positional consistency across ∂C_i with the neighboring regions. Vertices in the buffer zone B_i deform together with M_i, and propagate to M_i's neighboring regions for their deformation in next iteration.

Atlas and Transitions among Local Charts. Deformation of M_i (C_i) is performed under its local coordinate system F_i. Each vertex $v \in M_i$ stores its local coordinates with respect to F_i. If we denote the initial global coordinate system using a reference frame F_r, then F_i can be described using a rotation from F_r (No translation is introduced between coordinate systems). Hence, in the following, we also use the symbol F_i to denote the coordinate transformation from F_r. For example, given a point $\mathbf{x}$ originally considered in F_r, $F_i \circ \mathbf{x}$ indicates its coordinates after being transformed into the coordinate system of F_i. During the j-th iteration, we first solve a re-orientation $\hat{R}_i^j$ for M_i, and update F_i accordingly, $F_i \leftarrow \hat{R}_i^j \circ F_i$. Then, to construct C_i in the coordinate system of F_i, we transform vertices in buffer zone B_i from their own local coordinates into F_i, namely, $\forall \mathbf{x} \in M_k$ (represented under F_k), we get $\widetilde{\mathbf{x}} = F_i \circ F_k^{-1} \circ \mathbf{x}$. Now, $\widetilde{\mathbf{x}}$ is in the coordinate system of F_i. After local poly-square deformation, $\phi_i(\widetilde{\mathbf{x}})$ should be updated in its own coordinate system F_k after being transformed back to it,

i.e., $\mathbf{x} \leftarrow F_k \circ F_i^{-1} \circ \phi_i(\widetilde{\mathbf{x}})$. In Fig. 8, we demonstrate the pipeline of the updating of a local chart and its propagation.

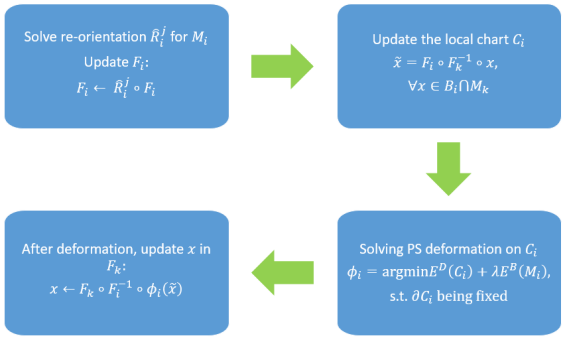


Figure 8: Update a local char and its propagation.

PPM Algorithm. Algorithm 1 summarizes our PPM algorithm. The deformations (Steps 5-12) are performed iteratively. In each iteration, we go through each subgroup G_k and deform the independent subregions simultaneously. Deformations are performed on all the deformation charts $C_j = M_j \bigcup B_j$. After each deformation, M_j is mapped to an updated poly-square and elements in B_j is modified and updated in their patches. In Step 13, a threshold T is used to check whether the deformation converges: when no vertex's movement is bigger than T during the last iteration, the algorithm stops. In our experiments, we set $T = 10^{-2}$, and the algorithm usually converges after around 15 iterations.

Algorithm 1 PPM Algorithm.

1: **In:** Subregions $\{M_j\}$, where $\bigcup M_j = M$;
2: **Out:** Poly-square Parameterization of M, i.e., $\phi_j = \Omega_j \rightarrow M_j$ subject to consistency constraints;
3: Divide $\{M_j\}$ into K disjoint groups $\{G_k\}$, each group contains independent (non-adjacent) subregions;
4: For each M_j, grow its buffer-zone B_j and get its deformation chart $C_j = M_j \bigcup B_j$;
5: **for** $k = 1$ to K **do**
6: Deform all the $C_j \in G_k$ in parallel:
7: Solve optimal re-orientation $\hat{R}_j$ (Prob. (11));
8: Update F_j: $F_j \leftarrow \hat{R}_j \circ F_j$;
9: Rotate M_j: $\mathbf{x} \in M_j, \mathbf{x} \leftarrow \hat{R}_j \circ \mathbf{x}$;
10: Build C_j: For $\forall \mathbf{x} \in B_j \bigcap M_r$, $\widetilde{\mathbf{x}} = F_j \circ F_r^{-1} \circ \mathbf{x}$;
11: Solve map ϕ_j (Prob. (14));
12: Propagate: $\mathbf{x} \leftarrow F_r \circ F_j^{-1} \circ \phi_j(\widetilde{\mathbf{x}})$, $\forall \mathbf{x} \in B_j \cap M_r$;
13: If Converge, STOP; Otherwise, GOTO STEP 5.

5.3 Adaptive Singularity Refinement

While PPM introduces some interior singularities, we can also further refine mesh locally when needed. Inspired by [47], we propose a new local singularity refinement method for regions that have large distortion or where singularities were inappropriately placed.

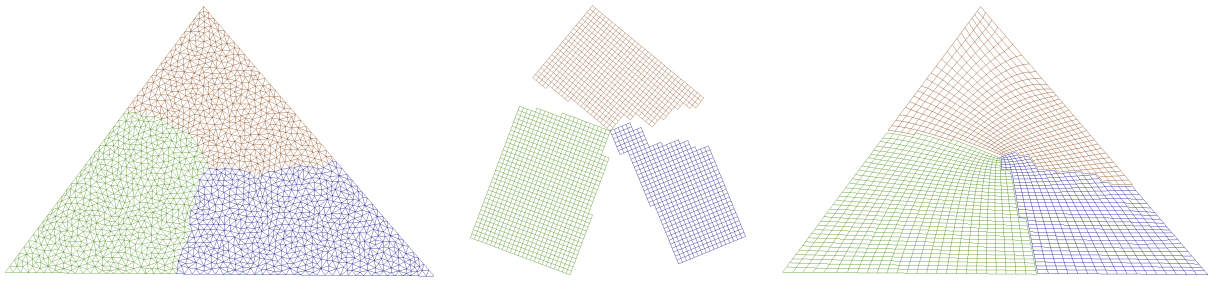
Poly-square maps generate two types of corners in the parametric domain, namely, 90° and 270° corners. If such corners are mapped to a flat boundary or feature curve, it will introduce big distortion and cannot be suppressed through variational relaxation. Fig. 12 (a) (blue points in pink patches) illustrates an example of such corners. We develop a set of topological strategies to modify such patches.

If we detect a vertex on boundary or feature curves which is 2-valent, then it means a 90 deg corner in the poly-square domain has been flattened here. Usually a **Triangle Patch** can be used to modify the patch surrounding such 90° corner. See Fig. 12(a): each pink region contains a triangle patch. When large distortion is detected in this region, it is because a 90° corner is mapped to a flat boundary region of R. Hence, such a region can be treated as a topological triangle (considering the arc on the boundary as one edge). We can quadrangulate this patch by the following algorithm.

As shown in Fig. 10, to quadrangulate a triangle patch, we just need to find a partition for the vertices on each edge. To form a valid quad tessellation, we need to assure the interval matching for this triangle patch. This leads to the following system:

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ b_0 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} N_0 \\ N_1 \\ N_2 \\ 0 \\ 0 \\ 0 \end{bmatrix} \leftrightarrow \begin{bmatrix} a_0 + b_0 = N_0 \\ a_1 + b_1 = N_1 \\ a_2 + b_2 = N_2 \\ a_0 - b_1 = 0 \\ a_1 - b_2 = 0 \\ a_2 - b_0 = 0 \end{bmatrix}$$

If there exists an integer solution, then this triangle patch can be quadrangulated with a 3-valent singularity inside that patch as shown in the left figure. If there is not an integer solution for it, we will quadrangulate it similar to the way we mesh the triangle region in Fig. 9, we can perform an extra partition in this local region. Specifically, we just need to run a 3-way partitioning on this region, and by balancing loads (region size), separator lengths, and geometry, we will always get a partition like (a) in Fig. 9. These three newly obtained subregions can then be processed like all other subregions using the PPM algorithm. A singularity point will be generated in the middle while the boundary corner is removed. Fig. 12(b) illustrates the result of this operation applied to the two pink



(a) (b) (c)

Figure 9: Quad Meshing of a Triangular Region. (a) The original mesh and the partitioning, (b) constructed poly-square of each sub-region, (c) meshing result.

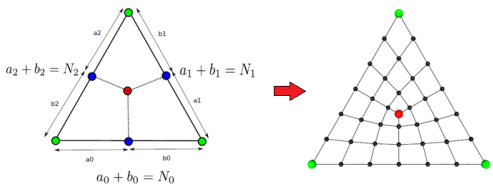


Figure 10: Quadrangulation a Triangle Patch

regions of (a). The distortion in this local area can be greatly reduced.

Another case is a vertex on boundary or feature curves, which is 4-valent. Then it means a 270 deg corner in the poly-square domain has been flattened here. Then a **Pentagon Patch** can often be traced around the 270° corner. See Fig. 12(c): each blue region contains a pentagon patch. Large distortion is detected in this region as well. This region can be treated as a topological pentagon. Similarly, with triangle patch, we can quadrangulate this patch by the following algorithm.

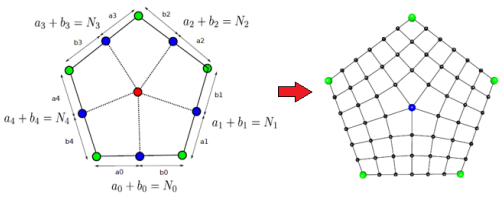


Figure 11: Quadrangulation a Pentagon Patch

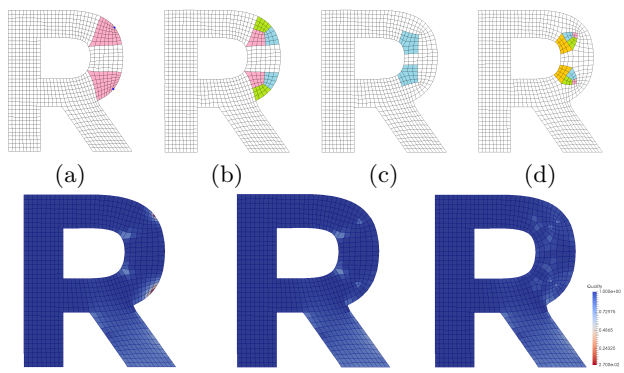
As shown in Fig. 11, to quadrangulation a pentagon patch. We just need to find a partition for the vertices on each edge. And we need to assure the interval matching for this pentagon patch. This leads to the following system:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} N_0 \\ N_1 \\ N_2 \\ N_3 \\ N_4 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\leftrightarrow \begin{bmatrix} a_0 + b_0 = N_0 \\ a_1 + b_1 = N_1 \\ a_2 + b_2 = N_2 \\ a_3 + b_3 = N_3 \\ a_4 + b_4 = N_4 \\ a_0 - b_3 = 0 \\ a_1 - b_4 = 0 \\ a_2 - b_0 = 0 \\ a_3 - b_1 = 0 \\ a_4 - b_2 = 0 \end{bmatrix}$$

If there exists an integer solution, then this pentagon patch can be quadrangulated with a 5-valent singularity inside that patch as shown in the left figure. If not, we can treat it with our more general Polygon patch quadrangulation algorithm.

A **Polygon patch** can always be traced as a $(n+2)$-sided topological polygon on the parametric domain (by iteratively performing $(n+2)$ times of moving forward then turning to the other iso-line counterclockwise). Then on this $(n+2)$-sided topological polygon we also perform a k-way partitioning, where we iteratively try integer k from $\lceil \frac{n}{2} \rceil$ to $(n+2)$. The intuition behind this range is that we will try different combinatorial partitions on this $(n+2)$-sided polygon: the lower bound $\lceil \frac{n}{2} \rceil$ is the subregion number that we will get if we keep removing 4-sided polygons from the region until we can no longer do it; while $(n+2)$ indicate a topological Catmull-Clark subdivision which directly splits this $(n+2)$-gon into $n+2$ quads. For example,



(e)0.97/0.32/0.07 (f)0.97/074/0.05 (g)0.98/0.84/0.05

Figure 12: Adaptive refinement of boundary singularities. The quality shown in second row is scaled Jacobian (average/minimum of Scaled Jacobian/standard deviation). (a) The PPM result without any singularity modification: each pink patch contains a Type-1 boundary singularity (Type-1 corner); (b) Refining the type-1 singularity; (c) Blue patch contains a Type-2 boundary singularity (Type-2 corner). (d) Refining the type-2 singularity. (e) The meshing result of GPM. (f) The meshing result of PPM without singularity modification. (g) The meshing result of PPM after singularity modification.

when $n = 4$, we have a topological hexagon. We first try the $\lceil \frac{n}{2} \rceil = 2$-way partitioning (a hexagon could split into 2 quad patches), then try 3, 4, till 6-way partitioning. When $n = 3$ we get a pentagon, and we will try $k \in [2, 5]$. Among all the partitioning results generated by different k, we can pick the one with the maximized minimal angle as our partition. Since n is usually very small, this selection is very quick. Fig. 12(d) illustrates the result of the modification on Type-2 corners in the blue regions in (c). In practice, users can get involved in this step to interactively select the template to repartition this local region and modify its singularity distribution. More complicated topological operations such as [47] can also be adopted.

6. EXPERIMENTAL RESULTS

We compare our algorithm with the advancing front algorithm [48] and the Global Poly-square Map (GPM) meshing discussed in Section 5.1. The advancing front generates unstructured meshes; GPM generates full structured meshes, while our algorithm, denoted as Feature-aligned Piecewise Poly-square Map (FPPM), generates semi-structured meshes.

The meshing distortion is measured using three metrics: *Scaled-Jacobian*, *Skewness*, and *Edge Ratio*.

The *scaled Jacobian* $\eta_f(Q)$ of each quad cell Q is calculated based on [49]:

$$\eta_f(Q) = \frac{1}{4}(\eta(x_1, Q) + \eta(x_2, Q) + \eta(x_3, Q) + \eta(x_4, Q)), \tag{15}$$

where $\eta(x, Q)$ is the scaled Jacobian distortion of vertex x in cell Q. $\eta_f \in (0, 1]$ for a valid quad cell, where 1 is ideal. $\eta_f < 0$ indicates a flipped cell.

The *Skewness* κ defined on a quad cell measures the deviation from the 90° angle:

$$\kappa(Q) = \max[\frac{\theta_{max} - 90}{90}, \frac{90 - \theta_{min}}{90}], \tag{16}$$

where θ_{max} and θ_{min} are the maximal and minimal interior angles in quad Q. $\kappa \in [0, 1]$ for a convex quad cell, where 0 is optimal. If $\kappa > 1$ the quad cell becomes concave.

The *Edge Ratio* r is the ratio of the longest edge length to the shortest one in cell Q, which measures the cell's deviation from a equilateral quad cell,

$$r(Q) = \frac{l_{max}}{l_{min}}, \tag{17}$$

where l_{max} and l_{min} are the maximal and minimal edge lengths in cell Q. $r \in [1, \infty)$ where 1 is optimal.

Using either one of the above 3 metrics, we can measure all the quad cells, and document the average and worst values, and the standard deviation. For example, we use the triplet $(\bar{\eta}, \eta_w, \sigma_\eta)$ to indicate the average, minimum (worst), and standard deviation of the scaled Jacobian, and $(\bar{\kappa}, \eta_\kappa, \sigma_\kappa)$ to indicate the average, maximum (worst), and standard deviation of skewness.

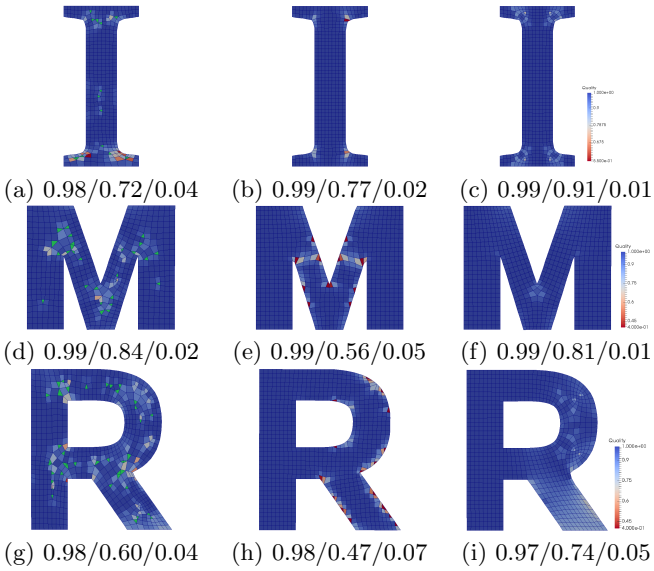
6.1 Comparing Meshing Quality on Small Geometries

Meshing Letters "IMR". Fig. 13 shows the quad meshing on three letters, I, M and R. The input is these letters' boundary contours extracted from images. The FPPM meshing results significantly outperform those generated by the other two algorithms.

Table. 1 reports the numerical measures computed on meshing results generated on small testing geometric regions. Besides the measure on scaled Jacobian $((\bar{\eta}, \eta_m, \sigma_\eta))$, we also report the triplets calculated on skewness and edge ratio, namely, $(\bar{\kappa}, \kappa_m, \sigma_\kappa)$ and $(\bar{r}, r_m, \sigma_r)$. Besides distortion, we also report the number of singularities N_s. Because the advancing front algorithm [48] also generates triangular cells in the final quad meshes, we also calculate the number of triangular elements and denote it as N_t. In all our GPM and FPPM results, N_t are zero. From these statistics, we can see that our PPM algorithm generates significantly less distorted (especially on skewness and edge

Table 1: Mesh Quality Comparison among the Advancing Front, GPM (Global Poly-Square Mapping), and PPM (Piecewise Poly-Square Mapping) algorithms. N_t is the number of triangle cells; N_s is the number of singularities; k is the sub-regions number, for advancing front, we solve the whole region directly without paritioning; t_p / t_m are the running time of partitioning and the running time of meshing in seconds; $\bar{\eta}^*/\eta_w^*/\sigma_\eta^*$ are the average, minimum, and standard deviation of Scaled Jacobian before applying Refinement; $\bar{\eta}/\eta_w/\sigma_\eta$ are the average, minimum, and standard deviation of Scaled Jacobian; $\bar{\kappa}/\kappa_w/\sigma_\kappa$ are the average, maximum, and standard deviation of Skewness; and $\bar{r}/r_w/\sigma_r$ are the average, maximum, and standard deviation of edge ratio.

Model	Meshing Method	# Cells	N_t	N_s	k	t_p/t_m	$\bar{\eta}^*/\eta_w^*/\sigma_\eta^*$	$\bar{\eta}/\eta_w/\sigma_\eta$	$\bar{\kappa}/\kappa_w/\sigma_\kappa$	$\bar{r}/r_w/\sigma_r$
I	Advancing Front	407	15	19	-	-/0.323	0.98/**0.72**/0.04	0.98/0.72/0.04	0.18/0.61/0.10	1.20/2.22/0.22
	GPM	477	0	4	10	0.085/0.728	0.99/0.71/0.02	0.99/0.77/0.02	**0.16**/0.73/**0.05**	**1.11**/2.36/0.17
	PPM	509	0	12	10	0.085/0.918	**0.99**/0.69/**0.02**	**0.99**/**0.91**/**0.01**	0.19/**0.57**/0.10	1.16/**1.92**/**0.16**
M	Advancing Front	550	27	37	-	-/0.141	**0.99**/**0.84**/**0.02**	0.99/**0.84**/0.02	0.15/**0.54**/0.11	1.24/2.95/0.20
	GPM	686	0	16	10	0.081/0.529	0.95/0.31/0.07	0.99/0.56/0.05	**0.10**/1.00/0.14	1.13/3.30/0.24
	PPM	744	0	1	10	0.081/0.736	0.97/0.48/0.09	**0.99**/0.81/**0.01**	0.13/0.75/**0.09**	**1.13**/**1.72**/**0.11**
R	Advancing Front	677	42	71	-	-/0.141	0.98/**0.60**/**0.04**	0.98/0.60/**0.04**	0.18/1.00/0.17	1.43/6.47/0.40
	GPM	818	0	28	10	0.081/0.529	**0.98**/0.42/0.07	**0.98**/0.47/0.07	**0.12**/1.00/0.28	**1.15**/4.31/0.24
	PPM	832	0	8	10	0.081/0.736	0.97/0.32/0.07	0.97/**0.74**/0.05	0.15/**0.47**/**0.13**	1.20/**4.02**/**0.23**



(a) 0.98/0.72/0.04 (b) 0.99/0.77/0.02 (c) 0.99/0.91/0.01

(d) 0.99/0.84/0.02 (e) 0.99/0.56/0.05 (f) 0.99/0.81/0.01

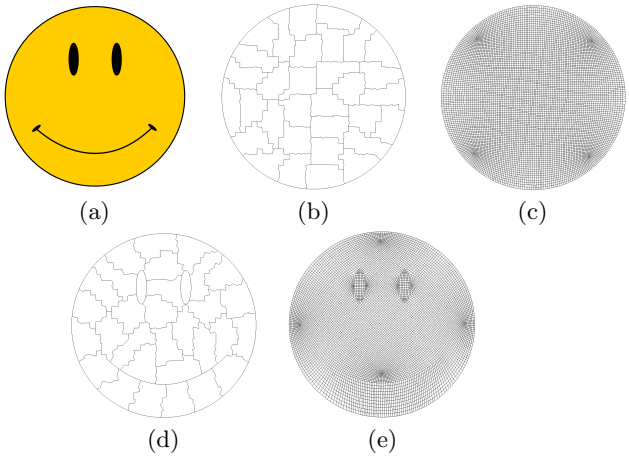
(g) 0.98/0.60/0.04 (h) 0.98/0.47/0.07 (i) 0.97/0.74/0.05

Figure 13: Meshing results of letters I, M and R. Quad meshing results generated by the advancing front, GPM, and PPM algorithms are illustrated in the left, middle, right columns respectively. $(\bar{\eta}, \eta_w, \sigma_\eta)$ are reported in captions accordingly.

ratio) meshes than advancing front and GPM algorithms. For the generated quad meshes, we define a vertex to be a *singularity* if it is (1) a boundary vertex but is not valence-3, or (2) an interior vertex but is not valence-4. Note that FPPM places the singularities according to geometry, while the GPM constructs a big poly-square with many boundary singularities. Thus, FPPM often has fewer singularities.

6.2 Meshing with Feature Alignment

Meshing a Smile Face. Fig. 14 shows the meshing of a smiling face. The feature lines include the contours of the two eyes and the mouth curve, as shown in (a). The partitioning and meshing result without considering these features are shown in (b) and (c), while results preserving these feature lines are shown in (d) and (e). Here feature preserving is fully enforced during the partitioning stage.



(a) (b) (c)

(d) (e)

Figure 14: Meshing for a smiling face. (a) The smiling face image; (b) The partitioning result without feature; (c) Meshing without feature; (d) Feature aligned partitioning; (e) Feature preserved meshing

Meshing a Turtle Shell. Fig. 15 illustrates the meshing result of a turtle shell image. We extract a turtle shell's boundary contour and some feature curves from this image (a) and uniformly sample it using 400 points. The partitioning and meshing result without considering these features are shown in (b) and (c), while the feature-preserved results are shown in (d) and (e). Feature alignment is fully resolved.

Runtime Statistics. We run our algorithm in sequential, 2-core, 5-core, 10-core, and 20-core scenarios to evaluate its scalability. The results are shown in Table 2.

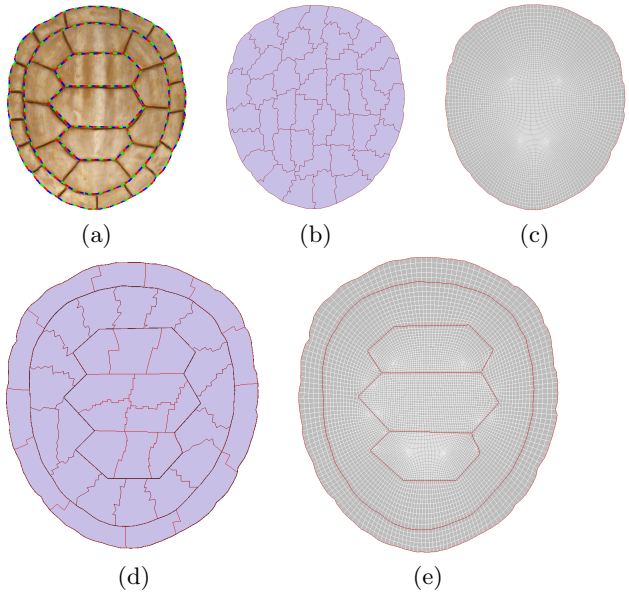


Figure 15: The quad meshing of a turtle shell model. (a) Features on the model, (b) data partitioning without considering features, (c) PPM meshing result without considering features, (d) data partitioning with feature considered, and (e) PPM meshing result with feature considered.

Table 2: Running Time in seconds. t_p / t_m are the partitioning and meshing time, respectively.

	Sequential	2-core	5-core	10-core	20-core
Smile Face	0.881/10.34	0.491/5.807	0.169/2.101	0.094/1.162	0.055/0.647
Turtle Shell	1.138/15.84	0.624/8.024	0.242/3.242	0.126/1.638	0.070/0.842

6.3 Meshing for large-scale Geometry

We evaluate the FPPM algorithm on big dataset obtained in practical modeling and scientific computing tasks. An example is a subsection of the Mississippi river as shown in Fig. 16(a). The levees and river boundary are treated as feature lines in the mesh generation. Using 50 cores, FPPM partitions this domain into 256 sub-regions in 113 seconds, and generates the quad mesh in 439 seconds. The final mesh contains $17,382,945$ elements with a minimum scaled Jacobian of 0.34. We can see the color-encoded result of the Scaled Jacobian in Fig. 16(b). In Fig. 16(c), a portion of the meshing result is zoomed in for closer visualization: the features are preserved very well.

6.4 Simulation Experiments

We solve partial differential equations on different meshes to compare these meshes' effectiveness in scientific computing. We compare the simulation efficiency using the convergence of Error Residual on three types of meshes: (1) triangle meshes, (2) unstructured quad meshes generated by advancing front, (3) semi-structured quad meshes generated by our FPPM.

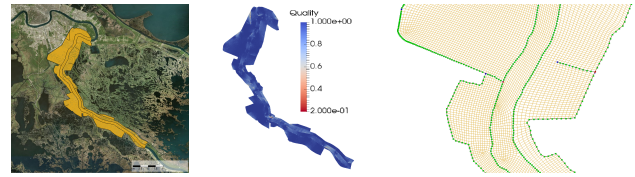


Figure 16: Quad meshing of the Mississippi river subsection and its levees. (a) The river subsection on Google Earth, (b) Color-encoded Scaled Jacobian of the mesh, and (c) a zoom-in meshing result.

6.4.1 Heat Diffusion Simulations

We solve a heat diffusion (the Laplacian Equation) $\frac{\partial T}{\partial t} + \nabla(D_t \nabla T) = 0$ where T is the temperature, and coefficient D_t is the thermal conductivity, on a mechanical part model as shown in Fig. 17(a): the red circle curve is the feature contour to be preserved in the final mesh. Initially, the outer boundary temperature is set as $273K$ (Kelvin), the interior area is set as $0K$. The temperature on the boundary and the center feature circle is fixed to $273K$ and $0K$ respectively. Fig. 17 shows the iteration-residual plot from the simulation. From the result statistics shown in Table. 3, we find that (1) The error residuals from the simulations performed on the quad meshes are one to two orders of magnitude smaller than that on the triangle meshes. (2) between the two quad meshes examined in this paper, our PPM algorithm produces significantly smaller error residuals in simulations.

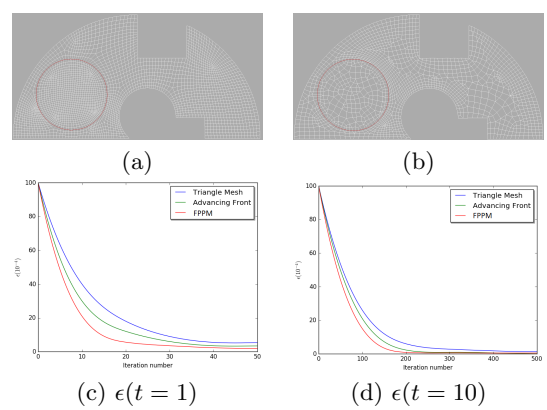


Figure 17: The iteration-residual plot of solving a Laplacian on three meshes generated from a Mechanical part model, at different t. (a) and (b) show show the meshing results from FPPM and the advancing front, respectively. (c) and (d) show the Iteration-residual at $t = 1$ (c) and $t = 10$ (d).

Table 3: Simulation on mechanical part with the Delaunay triangle meshes, quad meshes generated by advancing front, and by our PPM algorithm.

	# Cells	N_s	η_w	$\epsilon(t=1)$	$\epsilon(t=10)$
Triangle Mesh	2.1k	-	-	5.4 $\times 10^{-4}$	1.3 $\times 10^{-4}$
Advancing Front	1.4k	122	0.63	3.4 $\times 10^{-4}$	0.5 $\times 10^{-4}$
PPM	1.9k	12	0.66	1.9 $\times 10^{-4}$	0.5 $\times 10^{-4}$

6.4.2 Coastal Flow Simulations

One application of this meshing technique is to generate high-resolution best-fitting meshes in riverine, coastal, and estuarine areas for hydrodynamic simulations. To improve model accuracy and avoid numerical errors, a qualified mesh needs to fit some special features for the area of interest, such as coastlines, local bathymetries, hydraulic or marine structures. The normal structured (curvilinear) meshes are not suitable to fit sophisticated features in these areas, such as coastlines, river banks, small lakes, canals, levees, and breakwaters. There is a need to generate high-quality feature-preserved meshes for effective coastal simulations.

We set up an idealized case to demonstrate the advantage of this meshing technique. The process-based numerical model package, the Delft3D model by Deltares (http://www.deltares.nl/en/), is utilized for this hydrodynamic simulation. The Delft3D model suite by Deltares [35], which has been widely used in the world, is capable of simulating flows, sediment transport, waves, water quality, morphological changes and ecological processes in coastal, riverine and estuarine areas [50].

In our experiment, there is a flow channel ($80 \times 200 \times 5$ m) with double thin dams in it. The direction between the dam and the channel is 10 degrees. Two meshes with same sizes (80×200) are generated for comparison. One is a quad mesh that does not consider the feature of dams in Fig. 18(a). The other our generated quad mesh that fits double thin dams in Fig. 18(b). In our mesh, the double dams are sampled exactly by two straight lines in Fig. 18(d) and (f); while in the uniform mesh, the dams can only be represented approximately by two "zigzagged" lines in Fig. 18(c) and (e). Constant current conditions (1m/s) are applied at both ends of the flow channel. The result of flow velocity using our generated mesh in Fig. 18(d, f) which has the zoom-in details in Fig. 18(h) shows a smoothing distribution between double thin dams, which is reasonable. On the other hand, the result using the uniform mesh in Fig. 18(c, e) which has the zoom-in details in Fig. 18(g) shows numerical errors at the corners of "zigzagged" lines, which is unreal. We can see some "velocity arrows" are blocked because of the "zigzagged" lines and their directions are distorted. They are fake results which will not occur in the real world. These indicate the advantage of our meshing technique for hydrodynamic simulations. We can see the results in Fig. 18.

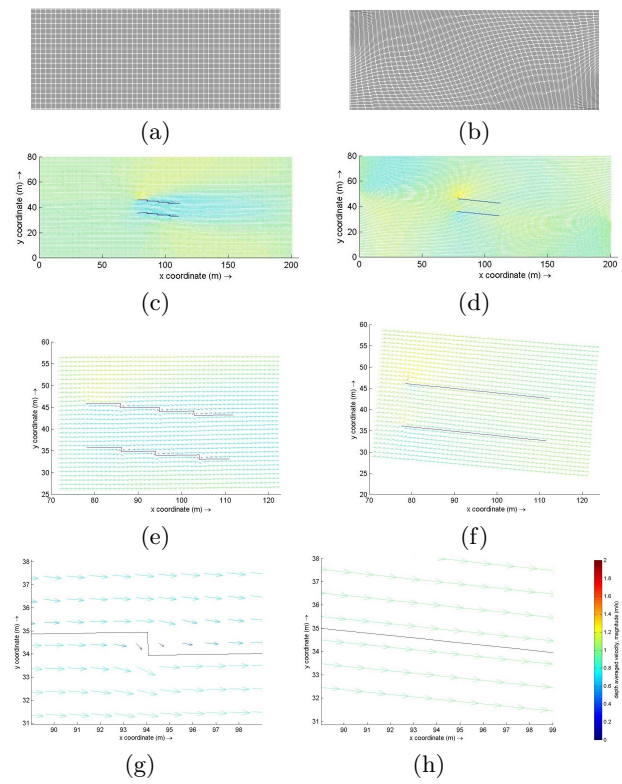


Figure 18: A flow simulation on a rectangle channel with double thin dams (features) in it. (a) The generated mesh without feature-preserving; (b) The feature-preserved mesh by considering the dam as the feature lines; (c, d) The simulated flow velocity fields using (a) and (b), respectively; (e, f) The zoom-in flow velocity fields around the dams from (c) and (d), respectively; (g, h) The more close zoom-in flow velocity filed for a small region from (c) and (d), respectively.

7. CONCLUSIONS

We present a new distributed computational framework to generate semi-structured quad meshes on large-scale 2D regions in parallel. This framework supports feature-preservation and interior singularity insertion. As a result, it can produce high-quality, low distortion meshes. We demonstrate the effectiveness of this meshing algorithm and its application to large-scale scientific simulations.

Limitations. One limitation of our generated semi-structured quad meshes is that the singularity placement and structured block layout are not globally optimized. Unnecessary singularities may be introduced during data partitioning. However, this cannot be easily solved due to the distributed nature of this algo-

rithm, where each local processor is not aware of the global layout and distant subregions' geometry. In this work, such a limitation is partly remedied through singularity refinement. In the near future, we will explore distributed block layout optimization and singularity placement.

8. ACKNOWLEDGMENTS

This work is supported by National Science Foundation IIS-1320959 and CyberSEES-1539567. Part of this work was finished when the first three authors were in Louisiana State University. We thank IMR reviewers for their valuable suggestions.

References

[1] Benzley S.E., Perry E., Merkley K., Clark B., Sjaardema G. "A comparison of all hexagonal and all tetrahedral finite element meshes for elastic and elasto-plastic analysis." *Proc. 4th Intl. Meshing Roundtable*, pp. 179–191. 1995

[2] Linardakis L., Chrisochoides N. "Delaunay decoupling method for parallel guaranteed quality planar mesh refinement." *SIAM JSC*, vol. 27, no. 4, 1394–1423, 2006

[3] Löhner R. "A 2nd generation parallel advancing front grid generator." *Proc. 21st IMR*, pp. 457–474. Springer, 2013

[4] Panitanarak T., Shontz S.M. "Mdec: Metis-based domain decomposition for parallel 2d mesh generation." *PCS*, vol. 4, 302–311, 2011

[5] Liu C., Yu W., Chen Z., Li X. "Distributed poly-square mapping for large-scale semi-structured quad mesh generation." *Computer-Aided Design*, vol. 90, 5–17, 2017

[6] Lo S. "Generating quadrilateral elements on plane and over curved surfaces." *Comp. & Structures*, vol. 31, no. 3, 421–426, 1989

[7] Lee C.K., Lo S. "A new scheme for the generation of a graded quadrilateral mesh." *Computers & structures*, vol. 52, no. 5, 847–857, 1994

[8] Potyondy D.O., Wawrzynek P.A., Ingraffea A.R. "An algorithm to generate quadrilateral or triangular element surface meshes in arbitrary domains with applications to crack propagation." *I. J. Numerical Methods in Engineering*, vol. 38, no. 16, 2677–2701, 1995

[9] Johnston B.P., Sullivan J.M., Kwasnik A. "Automatic conversion of triangular finite element meshes to quadrilateral elements." *IJNME*, 1991

[10] Blacker T.D., Stephenson M.B. "Paving: A new approach to automated quadrilateral mesh generation." *I. J. NME*, vol. 32, no. 4, 811–847, 1991

[11] Owen S.J., Staten M.L., Canann S.A., Saigal S. "Q-Morph: an indirect approach to advancing front quad meshing." *I. J. Numerical Methods in Engineering*, vol. 44, no. 9, 1317–1340, 1999

[12] Owen S.J., Saigal S. "H-Morph: an indirect approach to advancing front hex meshing." *IJNME*, vol. 49, no. 1-2, 289–312, 2000

[13] Remacle J.F., Henrotte F., Carrier-Baudouin T., Béchet E., Marchandise E., Geuzaine C., Mouton T. "A frontal Delaunay quad mesh generator using the L_∞ norm." *International Journal for Numerical Methods in Engineering*, vol. 94, no. 5, 494–512, 2013

[14] Kaelberer F., Nieser M., Polthier K. "QuadCover - Surface Parameterization using Branched Coverings." *CGF*, vol. 26, 375–384, 2007

[15] Bommes D., Zimmer H., Kobbelt L. "Mixed-integer Quadrangulation." *ACM Trans. Graph.*, vol. 28, no. 3, 77:1–77:10, 2009

[16] Bommes D., Campen M., Ebke H.C., Alliez P., Kobbelt L. "Integer-grid maps for reliable quad meshing." *TOG*, vol. 32, no. 4, 98, 2013

[17] Li Y., Liu Y., Xu W., Wang W., Guo B. "All-hex meshing using singularity-restricted field." *ACM TOG*, vol. 31, no. 6, 177, 2012

[18] Pellenard B., Orbay G., Chen J., Sohan S., Kwok W., Tristano J.R. "QMCF: QMorph Cross Field-Driven Quad-Dominant Meshing Algorithm." *Procedia Engineering*, vol. 82, 338–350, 2014

[19] Campen M., Kobbelt L. "Quad layout embedding via aligned parameterization." *CGF*, vol. 33, pp. 69–81. Wiley Online Library, 2014

[20] LYON M., CAMPEN M., BOMMES D., KOBBELT L. "Parametrization Quantization with Free Boundaries for Trimmed Quad Meshing." 2019

[21] Wang H., He Y., Li X., Gu X., Qin H. "Polycube splines." *Computer-Aided Design*, vol. 40, no. 6, 721–733, 2008

[22] Lin J., Jin X., Fan Z., Wang C.C. "Automatic polycube-maps." *International Conference on Geometric Modeling and Processing*, pp. 3–16. Springer, 2008

[23] Gregson J., Sheffer A., Zhang E. "All-Hex Mesh Generation via Volumetric PolyCube Deformation." *CGF*, vol. 30, pp. 1407–1416. 2011

[24] Yu W., Zhang K., Wan S., Li X. "Optimizing Polycube Domain Construction for Hexahedral Remeshing." *CAD*, vol. 46, 58–68, 2014

[25] Hu K., Zhang Y.J. "Centroidal Voronoi tessellation based polycube construction for adaptive all-hexahedral mesh generation." *Computer Methods in Applied Mechanics and Engineering*, vol. 305, 405–421, 2016

[26] Fang X., Xu W., Bao H., Huang J. "All-hex meshing using closed-form induced polycube." *TOG*, vol. 35, no. 4, 124, 2016

[27] Fogg H.J., Armstrong C.G., Robinson T.T. "Enhanced medial-axis-based block-structured meshing in 2-D." *CAD*, vol. 72, 87–101, 2016

[28] Rigby D. "Topmaker: A technique for automatic multi-block topology generation using the medial axis." *ASME/JSME 2003 4th Joint Fluids Summer Engineering Conference*, pp. 1991–1997. American Society of Mechanical Engineers, 2003

[29] Lévy B., Liu Y. "L_p Centroidal Voronoi Tessellation and its applications." *ACM Transactions on Graphics (TOG)*, vol. 29, p. 119. ACM, 2010

[30] Dong S., Bremer P.T., Garland M., Pascucci V., Hart J.C. "Spectral Surface Quadrangulation." *ACM Trans. Graph.*, vol. 25, no. 3, 1057–1066, 2006

[31] Zhang M., Huang J., Liu X., Bao H. "A Wave-based Anisotropic Quadrangulation Method." *TOG*, vol. 29, no. 4, 118:1–118:8, 2010

[32] Myles A., Pietroni N., Kovacs D., Zorin D. "Feature-aligned T-meshes." *ACM Transactions on Graphics (TOG)*, vol. 29, p. 117. ACM, 2010

[33] Jakob W., Tarini M., Panozzo D., Sorkine-Hornung O. "Instant field-aligned meshes." *TOG*, vol. 34, no. 6, 189, 2015

[34] Liu L., Zhang Y., Liu Y., Wang W. "Feature-preserving T-mesh construction using skeleton-based polycubes." *Computer-Aided Design*, vol. 58, 162–172, 2015

[35] Hydraulics D. "Delft3D-FLOW user manual." *Delft, the Netherlands*, 2006

[36] Tarini M., Puppo E., Panozzo D., Pietroni N., Cignoni P. "Simple quad domains for field aligned mesh parametrization." *TOG*, vol. 30, no. 6, 142, 2011

[37] Myles A., Pietroni N., Zorin D. "Robust field-aligned global parametrization." *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, 135, 2014

[38] Yu W., Li X. "A Geometry-aware Data Partitioning Algorithm for Parallel Quad Mesh Generation on Large-scale 2D Regions." *Proc. 24th IMR*, vol. 124, 44 – 56, 2015

[39] LaSalle D., Karypis G. "A parallel hill-climbing refinement algorithm for graph partitioning." *Parallel Processing (ICPP), 2016 45th International Conference on*, pp. 236–241. IEEE, 2016

[40] Roberts G.O., Rosenthal J.S., et al. "Optimal scaling for various Metropolis-Hastings algorithms." *Statistical science*, vol. 16, no. 4, 351–367, 2001

[41] Huang J., Jiang T., Shi Z., Tong Y., Bao H., Desbrun M. "l_1-Based Construction of Polycube Maps from Complex Shapes." *TOG*, vol. 33, no. 3, 25, 2014

[42] Hormann K., Greiner G. "MIPS: An efficient global parametrization method." *Curve and Surface Design*, pp. 153–162. 2000

[43] Fu X.M., Liu Y., Guo B. "Computing locally injective mappings by advanced MIPS." *TOG*, vol. 34, no. 4, 71, 2015

[44] Liu Y. "HLBFGS." 2009

[45] Fu X.M., Bai C.Y., Liu Y. "Efficient Volumetric PolyCube-Map Construction." *CGF*, vol. 35, pp. 97–106. Wiley Online Library, 2016

[46] Allwright J., Bordawekar R., Coddington P., Dincer K., Martin C. "A comparison of parallel graph coloring algorithms." *Technical report, Northeast Parallel Architectures Center at Syracuse University*, 1995

[47] Verma C.S., Suresh K. "αMST: A Robust Unified Algorithm for Quadrilateral Mesh Adaptation." *Procedia Engineering*, vol. 163, 238–250, 2016

[48] Autodesk-Inc. "Autodesk 3ds Max 2016." 2015

[49] Ruiz-Gironés E., Roca X., Sarrate J. "Optimizing mesh distortion by hierarchical iteration relocation of the nodes on the CAD entities." *Procedia Engineering*, vol. 82, 101–113, 2014

[50] Hu K., Ding P., Wang Z., Yang S. "A 2D/3D hydrodynamic and sediment transport model for the Yangtze Estuary, China." *Journal of Marine Systems*, vol. 77, no. 1-2, 114–136, 2009