Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Engineering 00 (2015) 000–000

Procedia Engineering

www.elsevier.com/locate/procedia

A Geometry-aware Data Partitioning Algorithm for Parallel Quad Mesh Generation on Large-scale 2D Regions

Wuyi Yu, Xin Li*

School of Electrical Engineering and Computer Science, Louisiana State University

Email: xinli@lsu.edu

Abstract

We develop a partitioning algorithm to decompose complex 2D data into small simple subregions for effective parallel quad meshing. We formulate the partitioning problem for effective parallel quad meshing as a quadratic integer optimization problem with linear constraints. Directly solving this problem is expensive for large-scale data partitioning. Hence, we suggest a more efficient two-step algorithm to obtain an approximate solution. First, we partition the region into a set of cells using L_∞ Centroidal Voronoi Tessellation (CVT), then we solve a graph partitioning on the dual graph of this CVT to minimize the total partitioning boundary length, while enforcing the load balancing and each subregion's connectivity. With this decomposition, subregions are distributed to multiple processors for parallel quadrilateral mesh generation. We demonstrate that our decomposition algorithm outperforms existing approaches by offering a higher-quality partitioning, and therefore, improved performance and quality in mesh generation.

Keywords: Geometry-aware Data Partitioning, Parallel Mesh Generation, Large-scale Structural Meshing.

1. Introduction

In recent years, the acquisition or generation of large high-resolution geometric datasets pose new challenges to the design of effective processing algorithms. These big and complex data are expensive to model and analyze using existing sequential algorithms, as the limited CPU and memory are often insufficient to handle billions of elements efficiently. Parallel algorithms utilizing high-performance computers make it possible to solve large and complex problems efficiently on hundreds or thousands of computing clusters and is therefore desirable.

Divide-and-conquer is a natural and effective strategy in parallel mesh generation for large geometric data. The given region or object is first decomposed into a set of solvable and simplified subparts; then each subpart is distributed to a working processor for mesh construction; finally, individually generated meshes are merged to get the final result. Parallel mesh generation strategies such as Delaunay-based methods and advancing front techniques have been used for both triangulation [1–3] or tetrahedralization [4,5]. Following this general paradigm, in this work we aim to develop a partitioning algorithm on comlex and large-scale 2D regions for parallel quadrilateral mesh generation.

In geometric processing through divide-and-conquer, the partitioning of data often directly dictates the algorithm's efficiency and quality. We formulate **three main criteria** as follows: (1) The areas of the subregions should be similar; (2) The boundary length of each subregion should be small compared with it's area; and (3) Each subregion should have desirable geometric property, and more specifically for quad meshing, it should have corner angles close to

$k\pi/2, k \in \{0, 1, 2, 3\}$. The **first two criteria** determine the parallel performance: the balance of work load on different processors, and the communication cost among processors. Efficient parallelization should balance the workload distributed to different processors and should minimize interprocessor communication to reduce synchronization and waiting. The **third criterion** on the subregion's geometry affects the quality of the final quad tessellation. For example, on a square subregion one can generate a quad mesh where every element is uniform and not sheared; but on a triangle subregion, the smallest angle of the resultant quad mesh will inevitably be smaller than the smallest boundary angle of this triangle patch. Hence, for effective quad mesh generation, it is desirable to partition the geometry into subregions whose boundary angles are near $k\pi/2$ to construct less-sheared quad elements. To incorporate the geometric constraint in data partitioning, however, is sometimes prohibitively expensive. We will discuss this issue in Section 3 and propose a more efficient strategy to tackle it.

After data partitioning, we distribute subregions to different working processors for mesh generation. In our implementation, we use consistent boundary sampling and *advancing front* for parallel meshing on each subregion. The sub-meshes are finally merged together then a relaxation is performed to get the final result. The *main contributions* of this paper include:

- We study geometry-aware data partitioning for effective parallel mesh generation, and suggest new models to partition large and complex 2D regions into subparts with desirable geometric shapes for quad meshing. Compared with existing partitioning algorithms, our approach could lead to more efficient parallel processing and higher-quality meshing results.
- We develop a parallel computing framework for quad mesh generation of large-scale 2D regions. It can effectively utilize parallel computational resources to handle big geometric data. We demonstrate an application in coastal modeling where massive-size coastal terrains and oceans need to be discretized for simulations.

2. Related Work

2.1. Data Partitioning

Given a geometric region M, a set of components $\{M_i\}$ is a *partition* of M if (1) their union is M, i.e., $\bigcup_i M_i = M$, and (2) all M_i are interior disjoint, namely, $\forall_{i \neq j} M_i^\circ \cap M_j^\circ = \emptyset$, where $M_i^\circ = M_i \backslash \partial M_i$ is the open set of M_i. Depending on the geometric processing applications, partitioning techniques consider different criteria accordingly. Thorough surveys on geometric partitioning algorithms have been given [6,7] for computer graphics and geometric modeling applications; some data benchmarks [8] have been built for evaluating these methods in these graphics applications.

Slightly different from the partitioning criteria considered in graphics applications, in order to obtain effective data partitioning for parallel computing, partition strategies can be classified into two categories: *extrinsic space partitioning* and *intrinsic manifold partitioning* methods.

We call the first strategy the *extrinsic space partitioning* method, because it partitions data by partitioning the data's embedding space. For example, data can be decomposed by spatial subdivision or partitioning structures such as quad-tree or octree [9], axis/planes [2], or blocks [10]. In parallel data processing literature, a very popular extrinsic space partitioning strategy is the *space filling curve* [11,12]. The idea is to first fill the N-dimensional space with a 1-dimensional curve and establish a bijective map between cells in the space and curve segments passing them. Different regions (cells) in the space are therefore indexed by this curve, and partitioning of the space (therefore, partitioning of data) is obtained by partitioning the curve accordingly. In general, data (space) partitioning using space-filling curves or other extrinsic space partitioning methods is very efficient, as demonstrated in several successful applications, such as computational physics, algebraic multigrid, PDE solving, adaptive mesh refinement [13,14]. However, algorithms based on spatial partitioning are not suitable to handle data that have complex geometry or nonuniform properties.

We call the second category the *intrinsic manifold partitioning* method, and it partitions the data model on its intrinsic tessellation. The data are discretized into a mesh or a graph, where elements or nodes are clustered into subparts directly [15] or recursively [16]. Among this category, an effective and widely used strategy is called *graph partitioning* [17–21], which usually produce good-quality partitions with balanced load and reduced communication. Solving the graph partitioning is NP-complete, and several effective strategies include spectral bisection [18], Kerninghan-Lin algorithm [17] and the multi-level scheme [19]. The spectral bisection algorithm [18] uses the spectral information to partition the graph. The eigenvector of the Laplacian matrix is computed to bisect the graph. Spectral Bisection usually produces a good partitioning, however it is very expensive to compute, especially for large matrix. On the other hand, the Kerninghan-Lin algorithm [17] is a fast heuristic scheme. It starts with an initial bipartition of the graph, then iteratively swaps a subset of vertices from each part to reduce the energy. This algorithm is fast, however, the initial partition is critical but often not easy to obtain. The multi-level method, including the widely adopted algorithm/software *METIS* [19], uses a three-phase scheme: first, the graph is simplified to a coarse graph; then a partitioning is performed on the coarsened graph; finally, the partitioning is modified during the progressive graph refinement. These existing graph partitioning algorithms focus on only tackling the load balancing and communication reduction issues. However, only considering these two criteria is insufficient. Incorporating extra geometric constraints is often critical in geometric modeling applications. The Medial Axis Domain Decomposition (MADD) algorithm [22] merges triangles to eliminate small angles then solves a graph partitioning on dual graph of the merged mesh. Subregions constructed using MADD partitioning usually possess larger corner angles than METIS. For our problem, it is desirable to have perpendicular corner angles. Therefore, a geometric term to incorporate this angle constraint can be formulated and included into the *graph partitioning* model. Furthermore, an additional connectivity constraint is needed to ensure each subregion form only one connected component. However, solving the original graph partitioning is already NP-hard, and the incorporation of these extra geometric constraints will further significantly increase the complexity of the problem (see Section 3 for details). This makes the efficient solving of this problem highly challenging.

2.2. Quadrilateral Mesh Generation

Quadrilateral mesh generation has been studied in computer graphics and geometric modeling fields. Quad meshes can be constructed through either the *indirect* or *direct* approaches. The *indirect* approaches first generate an intermediate structure/tessellation that can be easily constructed, e.g., a triangle mesh, then convert it into a quadrilateral mesh [23]. One big drawback of this method is that the layout of the unstructured elements in the intermediate tessellation determines the layout of final quad mesh, and there are usually a large number of singularities (i.e., non-valance-4 vertices) in the resultant quad meshes, which are usually undesirable for efficient simulation. The *direct* approaches construct quads on the model directly. Related techniques include quad-tree ([24]), template-guided decomposition ([25]) and advancing front([26]). A related problem is the quadrilateral mesh generation on curved surfaces. [27] gave a good survey on this topic. However, the curved geometry pose extra challenges in quad mesh generation and many recent surface quad meshing algorithms [28,29] use a cross frame field to guide the construction of the quad meshes. Finding an optimal cross frame field reduces to nonlinear integer programming, which is computationally expensive for large-scale geometric regions. Another related problem is the *quad layout patch* construction [30]. Its goal is to partition a surface into topologically rectangle patches, upon which quad meshes can be constructed. In this paper, we didn’t adopt this strategy, because the topological constraint on restricting subregions to be “4-sided polygons” is very expensive to enforce.

3. Region Partitioning

Data partitioning is the first step in a divide-and-conquer framework for parallel computational models. A good partitioning with balanced workload and small interprocess communication helps improve computational efficiency. Furthermore, in our parallel meshing problem, a good data partitioning is also critical in generating high-quality quad elements. We use three criteria to quantitatively evaluate a partitioning: **workload balance**, **total separator length**, and **separator angle deviation**. *Graph Partitioning* is a suitable strategy to solve our partitioning, because it can systematically model and optimize these criteria. In this section, we will first introduce the related notation (Sec. 3.1) and formulation of the three criteria (Sec. 3.2, 3.3 and 3.4) and the *Connectivity Constraints* (Sec. 3.5), then propose our algorithm in Sec. 3.6.

3.1. Notation

Given a tessellation $M = (V^M, E^M, F^M)$ of a 2D region, where V^M, E^M, F^M are the sets of vertices, edges, and faces (cells) respectively, let $G = (V^G, E^G)$ denotes its dual graph, where V^G, E^G are the node and arc sets respectively. A node $v \in V^G$ corresponds to a cell of M. Two nodes $v_1, v_2 \in V^G$ are connected by an arc if their corresponding cells are adjacent, namely, share an edge. Therefore, each arc of E^G also corresponds to an edge in E^M. The weight of a node $v \in V^G$ is defined to be the area of its associated cell $f \in F^M$, and an arc's weight is defined as the length of its associated edge. The partitioning on G can be computed on the dual graph G. A k-way *Graph Partitioning* divides V^G into k connected components, each of which is a subregion that will be processed individually. In practice, k-way partitioning is usually solved through recursive bisection [19]. Hence, we recursively partition G into two sub-graphs $G_0 = (V^{G_0}, E^{G_0})$ and $G_1 = (V^{G_1}, E^{G_1})$, where $V^{G_1} = V^G \setminus V^{G_0}$. This also partitions cells in the original tessellation M into two sets M_0 and M_1, if an edge $e \in M$ is shared by two cells f_i, f_j from distinct subsets, $f_i \in M_0$, $f_j \in M_1$, then edge e is called a **separator**.

For each node $v_i^G \in V^G$, we assign a variable x_i,

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

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

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

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 a $|E^G| \times |V^G|$ matrix. With these variables, we formulate the three criteria as follows.

3.2. Workload Balance

Load balancing refers to the practice of distributing approximately equal amounts of work among tasks, so that all tasks are kept busy all of the time. Unbalanced workload between working processors leads to inefficiency in parallel computing, as the slowest task often determines the overall running time. In our problem, the workload on each working processor can be estimated by the area of each subregion to mesh. On the dual graph, the subregion area can be calculated using the sum of weights of nodes in the corresponding subgraph.

A balanced partitioning should avoid big area difference between subregions. Hence, it is formulated as the following constraint:

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

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, $c_1 = c_2 = 0.1c$).

After a k-way partitioning is obtained, its workload balance can be evaluate by the ratio between the areas of the largest and smallest subregions:

$$R_W = A_{max}/A_{min}, \tag{2}$$

where A_{max} and A_{min} are the areas of the largest and smallest subregions respectively. R_W close to 1 means better workload balance.

3.3. Total Separator Length

In parallel computing, inter-process communication means overhead. A smaller total separator length usually indicates less communication cost. In our geometric data partitioning, a smaller separator length also indicates (1) smoother subregion boarder lines, (2) better geometric compactness of subregions, and (3) more efficient post-processing refinement (see Section 4). Therefore, it is desirable to minimize the *total separator length*

$$L_S = \mathbf{y}^T \mathbf{W}_e \mathbf{y} = \mathbf{x}^T \mathbf{U}^T \mathbf{W}_e \mathbf{U} \mathbf{x}, \tag{3}$$

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.

With the above two criteria, solving a graph partitioning can be formulated as:

$$\begin{aligned} \min \quad & \mathbf{x}^T \mathbf{U}^T \mathbf{W}_e \mathbf{U} \mathbf{x}, \\ \text{subject to} \quad & c_1 \le \mathbf{x}^T \mathbf{w}_v - c \le c_2, \\ & x_i \in \{0, 1\}. \end{aligned} \tag{4}$$

Optimizing problem (4) is NP-hard. For large data, multilevel schemes such as METIS [19] have been widely adopted to obtain good approximate solutions. Figure 1 illustrates a simple example. The input tessellation M is shown in (a) and the partitioning result by METIS [19] is shown in (b). The two partitioned subregions colored in blue and red respectively. We can see that the two subregions have the same area with total separator length minimized.

Minimizing the total separator length will makes the boundary of subregions straight, and it can increase the *compactness* of each subregion. On each subregion M_i, the *compactness* can be measured by the ratio between the boundary separator's length of M_i and the area of M_i. Globally, we can compute an *average compactness ratio*,

$$\hat{R}_C = \frac{L_S}{kA_M}, \tag{5}$$

where k is number of subregions, and A_M is the total area of the region. Smaller $\hat{R}_C$ comes from a smaller L_S and indicates better compactness.

3.4. Separator Angle Deviation

Solving the graph partitioning formulated in Eq. (4) will result in balanced area and minimized total separator length. However, in many geometric processing tasks, constraints on the geometry of subregions are important. In our quad meshing problem, ideally, each subregion should look like a square. Less strictly, since we use the advancing front technique to generate quad meshes (Section 4), it is desirable to have angles between separators close to $\frac{k\pi}{2}$. Therefore, we include a new separator angle term into graph partitioning to penalize each such angle's deviation from $\frac{k\pi}{2}$.

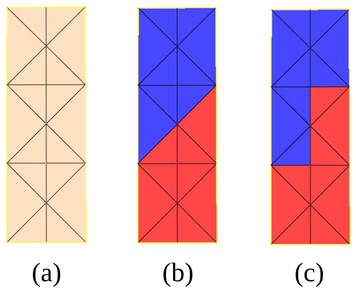


Fig. 1: Graph Partitioning on a simple example. (a) The original mesh; (b) the two (red and blue) subregions obtained by METIS [19]; (c) the partitioning result with separator angle deviation considered.

Angle Deviation Function. Consider the original tessellation M of the given 2D region, suppose two edges $e_i, e_j \in E^M$ share a vertex v and form an angle $\theta_{i,j}$. For concise representation, in the following, we use $Inc(i, j) = 1$ to denote e_i and e_j are incident, and $Inc(i, j) = 0$ means they are not incident. We define an *angle deviation function*

$$\delta_{\theta_{i,j}} = \begin{cases} \min_k\{|\theta_{i,j} - \frac{k\pi}{2}|, k \in \{0,1,2,3\}\}, & \text{if } Inc(i,j) = 1 \\ 0, & \text{if } Inc(i,j) = 0 \end{cases} \tag{6}$$

to describe the deviation from angle $\theta_{i,j}$ to the nearest $\frac{k\pi}{2}$ angle.

Accumulated separator angle deviation can then be formulated as

$$D_\theta = \mathbf{y}^T \mathbf{W}_\theta \mathbf{y} = \mathbf{x}^T \mathbf{U}^T \mathbf{W}_\theta \mathbf{U} \mathbf{x}, \tag{7}$$

where $\mathbf{y} = (y_{e_1}, y_{e_2}, \ldots, y_{e_n})^T$ is the edge variable vector, and $\mathbf{W}_\theta = \begin{pmatrix} 0 & \delta_{\theta_{1,2}} & \delta_{\theta_{1,3}} & \ldots & \delta_{\theta_{1,n}} \\ \delta_{\theta_{2,1}} & 0 & \ldots & \ldots & \delta_{\theta_{2,n}} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \delta_{\theta_{n,1}} & \ldots & \ldots & \ldots & 0 \end{pmatrix}$, $\mathbf{W}_\theta$ is an $|E^G| \times |E^G|$ matrix storing deviation angles $\delta_{\theta_{i,j}}$. This angle deviation matrix $\mathbf{W}_\theta$ can be precomputed by traversing all the edge pairs of the tessellation M once.

Furthermore, we can show that Eq. (7) *will evaluate and only evaluate the angle deviation between adjacent separators*. Suppose two edges $e_i = [u, v], e_j = [u, w]$ are incident separators, sharing vertex u. Then, (1) $y_{e_i} \neq 0$ and $y_{e_j} \neq 0$, and (2) v and w belong to the same subregion and have a same indicator value, $x_v = x_w$. From (1) and (2), it is not difficult to see that y_{e_i} and y_{e_j} are both -1 or both $+1$. Therefore, a *non-negative contribution* $\delta_{\theta_{i,j}}$ will be added to the accumulated separator angle deviation D_θ. To obtain a geometrically desirable partitioning, we can minimize the separator angle deviation term (7) together with total separator length (3), with the workload balance constraint (1).

Finally, after the partitioning, we can numerically evaluate the *average separator angle deviation*:

$$\hat{\delta}_\theta = \frac{1}{N_C}\Big(\sum_{Inc(i,j)=1} \delta_{\theta_{i,j}}\Big), \tag{8}$$

where N_C is the total number of corners formed by incident separators. The smaller $\hat{\delta}_\theta$ is, the better.

3.5. Connectivity Constraints

Currently we have formulated the three criteria. To guarantee the result is a bipartitioning, we need to impose the *connectivity constraint*, which is often not explicitly considered in existing graph partitioning literature, due to its complexity. Specifically, nodes in each subregion should form one connected component. Without explicitly enforcing this, although minimizing total separator length often tends to penalize the partition that produces multiple disjoint subsets, we can sometimes observe that more than one connected components exist in one subregion.

To enforce connectivity of each subregion, we can formulate the following explicit constraint. Given a (dual) graph $G = (V^G, E^G)$, for any pair of non-adjacent nodes u, v, we define a node set $S \subset V \setminus \{u, v\}$ to be a *node-cut set* that separates u and v, if there is no path between u and v after S is removed from G. For example, in the graph shown in Fig 2, for node pair $\{1, 4\}$, $\{2, 5, 7\}$ is a node-cut set and $\{3, 6, 8\}$ is another node-cut set. For $\{u, v\}$ that $[u, v] \notin E^G$, we define $\Gamma(u, v)$ to be the set consisting of all the node-cut sets of $\{u, v\}$. In this example, $\Gamma(1, 4) = \{\{2, 5, 7\}, \{2, 5, 8\}, \{2, 6, 7\}, \{2, 6, 8\}, \{3, 5, 7\}, \{3, 5, 8\}, \{3, 6, 7\}, \{3, 6, 8\}\}$.

The connectivity constraint can be described as: between each pair of nodes u, v that are in the same subregion, any node-cut set in $\Gamma(u, v)$ must have at least one node being assigned to this subregion. Using the binary variable x_i defined previously, the connectivity constraints can be formulated as a set of linear constraints. For any two non-adjacent nodes in subgraph G_1:

$$\sum_{w \in S} x_w \geq x_u + x_v - 1, \forall S \in \Gamma(u, v), \text{for } \forall x_u = x_v = 1, [u, v] \notin E^G, \quad (9)$$

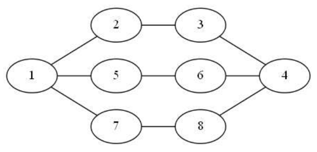


Fig. 2: An example graph for connectivity constraint.

meaning that every node-cut set must have at least one node being assigned as 1. Similarly, for any two non-adjacent nodes in subgraph G_0:

$$\sum_{w \in S} x_w \leq x_u + x_v + |S| - 1, \forall S \in \Gamma(u, v), \text{for } \forall x_u = x_v = 0, [u, v] \notin E^G. \quad (10)$$

These constraints ensure that there is at least one path connecting non-adjacent nodes u and v if they are grouped to a same subgraph.

By incorporating both the *separator angle deviation* (Eq. 6) and *connectivity constraints* (Eqs. 9,10) into graph partitioning, the quadratic integer programming problem with linear constraints can be solved through branch-and-bound algorithm. In our implementation, we use the open-source *Basic OpeN-source Mixed INteger* (BONMIN) solver from [31] to obtain a solution. To solve mixed integer non-linear programming problems, BONMIN allows one to choose optimization strategies including branch-bound, outer approximation (OA), Quesada Grossman branch-cut, and Hybrid OA based branch-cut.

Figure 1 (c) shows the solution of this new graph partitioning. With the minimization of separator angle deviation and the enforcement of connectivity constraints, a partitioning more suitable for quad mesh generation is obtained. However, because (1) incorporating separator angle deviation significantly increase the numbers of non-zero cross multiplication of indicator variables, and (2) the enforcement of connectivity introduces an exponential number of linear constraints. Solving such a new problem becomes prohibitively expensive for even moderately large problem.

3.6. Our Two-Step Partitioning Algorithm

We proposed a two-stage partitioning scheme to incorporate the two new constraints discussed in the last section. A key observation this idea based upon is that: *if the cells of the initial tessellation has near-square geometry, then the partition performed on the dual graph of this tessellation tends to have smaller separator angle deviation*. Hence, first, we use L_∞-CVT to generate a tessellation with cells similar to squares (Sec. 3.6.1); second, we solve our graph partitioning on this tessellation to get a set of subregions with balanced workload and small total separator length, with connectivity constraints enforced on the subgraphs during their refinement.(Sec. 3.6.2) We call this algorithm a L_∞-*CVT-GP* algorithm for short, and the idea is illustrated in Fig. 3.

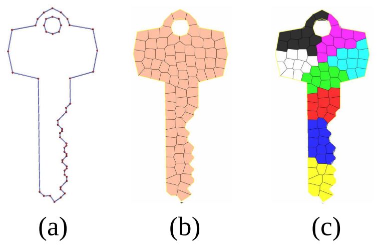


Fig. 3: Partitioning a 2D "Key" Region for Parallel Quad Mesh Generation. (a) The input 2D boundary; (b) L_∞-norm CVT on the input boundary; (c) our partitioning result, with different subregions indicated using different colors.

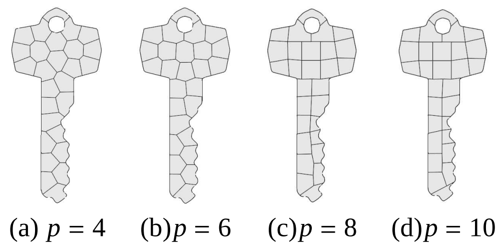


Fig. 4: L_p-CVT using different p values. The results are similar when $p >= 8$. In our experiments, we use L_8-CVT to approximate L_∞-CVT.

3.6.1. L_∞ Centroidal Voronoi Tessellation

The Voronoi diagram is a fundamental geometric structure widely used in various fields, especially in geometric modeling and computer graphics. A 2D *Voronoi Diagram* of a given set of distinct points $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^n$ in $\mathbf{R}^2$ is defined by a set of Voronoi cells $\{\Omega_i\}_{i=1}^n$ where

$$\Omega_i = \{\mathbf{x} \in \mathbf{R}^2 | \ ||\mathbf{x} - \mathbf{x}_i|| \leq ||\mathbf{x} - \mathbf{x}_j||, \forall j \neq i\}.$$

These points $\mathbf{X}$ are called *sites*. Each Voronoi cell Ω_i is the intersection of a set of half-planes. A *Clipped Voronoi Diagram* for the sites $\mathbf{X}$ within a given 2D domain Ω is the intersection of the Voronoi Tessellation and the domain Ω, denoted by $\{\Omega_i|_\Omega, i = 1, \dots, n\}$, where

$$\Omega_i|_\Omega = \{\mathbf{x} \in \Omega | \ ||\mathbf{x} - \mathbf{x}_i|| \leq ||\mathbf{x} - \mathbf{x}_j||, \forall j \neq i\}. \tag{11}$$

In other words, $\Omega_i|_\Omega = \Omega_i \cap \Omega$. The *Centroidal Voronoi Tessellation* (CVT) is that each site of the Voronoi cell is coincident to this cell's centroid.

The L_p Centroidal Voronoi Tessellation Energy [32] can be defined as:

$$F(\mathbf{X}) = F([\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]) = \sum_i \int_{\Omega_i \cap \Omega} ||\mathbf{y} - \mathbf{x}_i||_p^p d\mathbf{y} \tag{12}$$

where $||\mathbf{z}||_p = (\sum_j |z_j|^p)^{\frac{1}{p}}$, z_j is the j-th coordinate of a 2D point $\mathbf{z}$. Figure 5 shows the trajectory with different p. With the increase of p, the trajectory approximates to the square. The trajectory of L_∞ is a perfect square, where the L_∞ norm of a d-dimensional vector $\mathbf{z}$ is the maximal component in $\mathbf{z}$, $||\mathbf{z}||_\infty = \max_j |z_j|$. For efficient CVT computation, we choose a sufficiently large p to approximate the L_∞ norm. This also allows us to efficiently compute the gradient of $F(\mathbf{X})$ of Eq. (12). We test L_p-CVT on the key model using different p values. And as the results illustrated in Figure 4, when $p > 8$, the difference of L_p-CVT energy becomes very small, so we use L_8-CVT to approximate L_∞-CVT in all our experiments. Since we use L_8 which is smooth, the optimization of CVT energy F can be solved efficiently using the quasi-Newton BFGS solver [33].

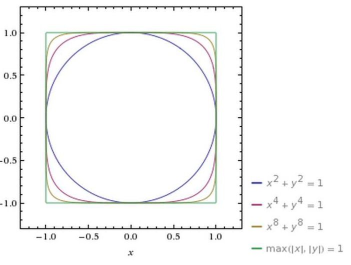


Fig. 5: The trajectory of $L_p(x, y) = 1$ with different p values. With the increase of p the trajectory gradually approximates the unit square. $L_\infty(x, y) = 1$ gives a square.

In summary, the algorithm to compute the L_p-CVT on the input 2D region Ω has four steps.

1. Get a uniformly sampled sites set $\mathbf{X}$. In our implementation, we simply embed Ω on a spatial grid, the grid points inside Ω are the initial sites.
2. Use the sweeping line algorithm [34] to construct the initial Voronoi Diagram.
3. Get the L_∞-CVT tessellation by optimizing CVT energy.
4. Clip the CVT using $\partial\Omega$ to get the L_∞-CVT of Ω.

3.6.2. Graph Partitioning Based on L_∞-CVT

After solving the L_∞-CVT, we get a tessellation $\widetilde{M}$ of the 2D region. Then on $\widetilde{M}$'s dual graph $\widetilde{G}$, we solve the graph partitioning formulated in Eq. (4). Following the heuristics used in [19], instead of explicitly enforcing the large number of connectivity constraints in Eqs. (9, 10), we can adopt a region growing examination to check the connectivity of each subregion, disconnected elements will be grouped into the other subregion automatically. During our optimization, after every K iterations, we perform such an examination and update on $\widetilde{G}$. In practice, this strategy is efficient and effective in enforcing the connectivity of subregions.

We test different partitioning results on the Key model using (a) original graph partitioning by METIS, (b) geometry-integrated graph partitioning solver introduced in Sections 3.4 and 3.5, and (c) this two-step L_∞-CVT-GP algorithm. We also evaluate the workload balancing ratio R_W, average compactness $\hat{R}_C$, and average separator angle deviation $\hat{\delta}_\theta$ on these partition results. Using these three partitioning methods, R_W are 1.13, 1.14, and 1.18, respectively; $\hat{R}_C$ are 1.11, 1.15, and 1.18, respectively, and $\hat{\delta}_\theta$ are 0.26, 0.12, and 0.15, respectively.

As expected, the graph partitioning without geometric constraint focuses on workload balance and separator lengths, and gets best R_W and $\hat{R}_C$, but very bad angle deviation. The expensive geometry-integrated graph partitioning produces smallest angle deviation. This two-step L_∞-CVT-GP produces results with slightly worse R_W, $\hat{R}_C$, and $\hat{\delta}_\theta$ than that from the geometry-integrated graph partitioning. But its separator angle deviation is significantly better than the original graph partitioning, and its speed is significantly faster than geometry-integrated graph partitioning. More thorough comparisons will be given in Section 5.

Through the L_∞-CVT, we can also directly obtain a region partitioning. Is such a partitioning sufficient, so that we no longer need to further solve a graph partitioning? This section illustrates that the two-step L_∞-CVT-GP algorithm usually leads to a better data partition for the parallel meshing problem. One observation is that: with the *increase of the number of Voronoi sites/cells, the decomposition from L_∞-CVT will become more uniform (better workload balancing) and more square-like (smaller separator angle deviation).* We have performed extensive experiments on L_∞-CVT to verify this. Table 1 is the partitioning statistics on the key and the pipe model. If we directly evaluate the quality of the partitioning suggested by the CVT decomposition, when the number of Voronoi cells increases from 16 to 128, the workload balancing ratio R_W (Eq. (2)) reduces from 1.65 to 1.23, while the average separator angle deviation $\hat{\delta}_\theta$ also reduces from 0.26 to 0.18.

Table 1: Partitioning on the Key and Pipe models using direct CVT versus using our CVT-GP algorithm. N_S is the number of subregions, R_W, $\hat{R}_C$, and $\hat{\delta}_\theta$ are the workload balance ratio, average compactness, and average separator angle deviation.

	Key			Pipe		
	CVT/CVT-GP			CVT/CVT-GP		
N_S	16	32	128	32	64	256
R_W	1.65/1.15	1.45/1.24	1.23/1.11	1.81/1.31	1.65/1.31	1.41/1.27
$\hat{R}_C$	1.21/1.11	2.21/ 2.01	9.41/9.24	1.82/1.77	4.04/3.88	18.11/17.84
$\hat{\delta}_\theta$	0.26/0.21	0.21/0.16	0.18/0.17	0.23/0.20	0.18/0.15	0.14/0.12

We have the following observations.

- For a k-way partition, direct partitioning through a L_∞-CVT decomposition with k cells will lead to a worse partitioning result than our two-step algorithm (which first generates n cells ($n > k$) then performs a graph partition to get k subregions).
- In order obtain a partition with similar R_W and $\hat{\delta}_\theta$, direct L_∞-CVT decomposition should use more sites. But this will increase the total separator length, resulting in bigger overhead and more singularities.

And we conclude that the L_∞-CVT-GP algorithm offers a better partition than the direct L_∞-CVT decomposition. More comparisons will be given in Section 5.

4. Quad Mesh Generation

4.1. Parallel Quad Meshing on Subregions

After the entire 2D region is partitioned, subregions can be sent to different processors for simultaneous mesh generation. Different quad meshing techniques can be applied on the sub-regions. One requirement is to make sure the neighbouring subregion boundaries should have consistent vertices. We use the **advancing front technique** [26] to tile the interior regions with quads. To ensure the individually constructed sub-meshes can be composed directly, we need to sample boundary vertices consistently on the shared edge of adjacent subregions. We use a simple sampling scheme: first, we compute an average edge length $\bar{l}$ from input boundary line segments, then on each interior partitioning boundary curve S we evenly sample $\frac{l_S}{2\bar{l}}$ points, where l_S is the length of S. To ensure the valid generation of a full quad mesh, the number of the points on the subregion boundary must be even [26], therefore, each interior separator line segments is subdivided into two to ensure the number of sampled boundary vertices is even.

The advancing front algorithm initiates a wave front from the boundary of each subregion, along which quadrangles are constructed inwards until all empty regions are tiled. We implement the advancing front following [26]. The wave front propagates until the front has 6 or fewer vertices, by when a template is used to finish the quad mesh generation. Readers are referred to [26] for details.

4.2. Post-processing after Composition

After composing meshes of subregions, we perform a Laplacian relaxation on vertices near separators to improve the smoothness of the orientations of mesh elements on the partitioning boundary. Each vertex moves towards the mass center of its neighboring vertices. Since our partitioning algorithm minimizes the total separator length, this relaxation only applies to a small number of vertices and takes a short time to process. In our experiments, up to 50 iterations are applied to each vertices within a five-ring buffer zone surrounding separators. In our implementation, we didn't parallelize this post-processing. But naturally, this refinement can be easily parallelized if needed.

5. Experimental Results

We perform experiments on our high performance computing clusters, SuperMIC, which consist of 128 computing nodes. Each node has two 2.6GHz 8-Core Xeon 64-bit Processors and 32GB memory. Five datasets have been tested: a key model with 21.6M boundary segments and 1 inner hole, a pipe model with 57.6M boundary segments and 9 holes, and three coastal ocean/terrain regions : the Gulf of Mexico, Matagorda Bay, and West bay, with 230M, 250M, and 550M boundary vertices, respectively. The generated meshes of the two bays and the Gulf of Mexico have about 3, 4.5, and 10 billion elements, respectively.

We compare algorithms in three aspects: (1) **Decomposition Quality**: the workload balance ratio, total separator length, and separator angle deviation. (2) **Parallel Computation Efficiency**: the running time on each working processors and the total speedup in quad meshing. (3) **Meshing Quality**: the scaled-Jacobian of quad elements and number of singularities of the final mesh.

Our experiments are designed to compare 4 decomposition methods: (1) Partitioning via direct L_∞-CVT, (2) Partitioning by METIS [19], a very widely used Graph Partitioning solver, (3) Partitioning by Medial Axis Domain Decomposition (MADD) [22], and (4) Our L_∞-CVT-GP algorithm.

5.1. Partitioning Quality Comparison

Figure 6 illustrates the four partitioning results on the key and pipe models. Table 2 gives the partitioning statistics for key and pipe model respectively. The workload balance ratio R_W is calculated following Eq. (2). When R_W is closer to 1, a better workload balance is achieved. The average compactness $\hat{R}_C$ is calculated following Eq. (5). The average separator angle deviation $\hat{\delta}_\theta$ is calculated following Eq. (8). Note that for each of these three terms, *the smaller the measured value is, the better*. From these experiments, we can see that:

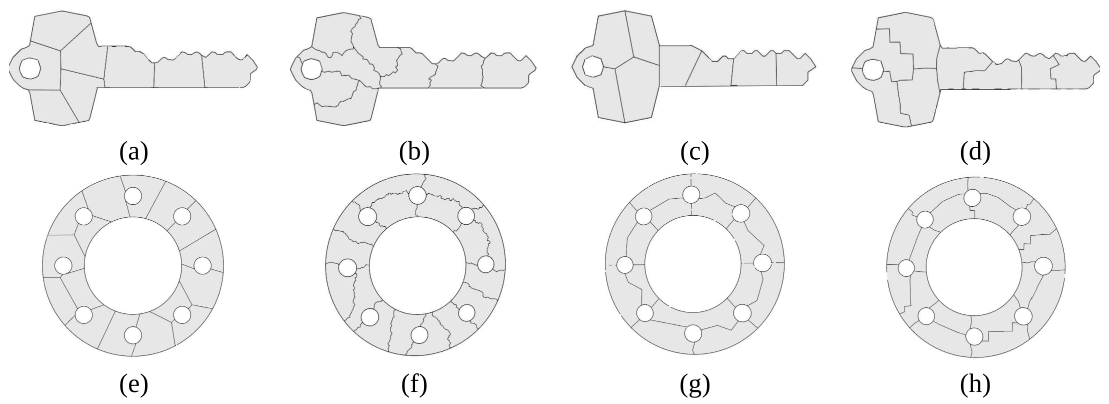


Fig. 6: (a-d) Partitioning the key model into 8 subregions using direct L_∞-CVT, METIS, MADD, and our L_∞-CVT-GP algorithm. (e-h) Partitioning the pipe model into 16 subregions using direct L_∞-CVT, METIS, MADD and our algorithm.

Table 2: Decomposition quality comparison (the key and pipe model): direct L_∞-CVT, METIS, MADD and our L_∞-CVT-GP method (initialized 4000 CVT Cells). N_S is the number of subregions. Our method has comparable workload balance and average compactness, while our $\hat{\delta}_\theta$ is up to about 50% and 55% smaller than the METIS method on key and pipe model respectively; our $\hat{\delta}_\theta$ is 20% and 15% smaller than MADD on two models respectively. Compared with L_∞ method, our R_W is 40% smaller.

Key	L_∞-CVT			METIS			MADD			Our Method (4000 cells)		
N_S	32	128	1024	32	128	1024	32	128	1024	32	128	1024
R_W	2.31	2.01	1.92	1.06	1.03	1.02	1.21	1.13	1.08	1.08	1.06	1.04
$\hat{R}_C$	3.93	13.25	104.3	3.61	12.21	93.31	3.98	14.26	104.91	3.85	13.57	105.21
$\hat{\delta}_\theta$	0.21/0.14	0.18/0.15	0.15/0.14	0.40/0.24	0.32/0.26	0.41/0.21	0.36/0.16	0.32/0.20	0.25/0.19	0.16/0.14	0.17/0.15	0.15/0.14

Pipe	L_∞-CVT			METIS			MADD			Our Method (4000 cells)		
N_S	32	128	1024	32	128	1024	32	128	1024	32	128	1024
R_W	2.39	2.09	1.98	1.07	1.05	1.04	1.27	1.25	1.13	1.14	1.13	1.10
$\hat{R}_C$	1.34	4.76	36.51	1.18	4.64	36.37	1.32	5.06	35.48	1.25	4.71	36.42
$\hat{\delta}_\theta$	0.22/0.14	0.18/0.15	0.15/0.15	0.41/0.23	0.32/0.25	0.42/0.21	0.36/0.15	0.32/0.20	0.25/0.21	0.18/0.12	0.15/0.12	0.13/0.11

1) For *workload balance*: METIS leads to the smallest R_W. The R_W of our method is about 5% bigger than METIS. R_W of MADD is about 10% bigger than ours. The direct CVT partitioning has worst workload balance and its R_W is about 55% bigger than ours.

2) For *average compactness* $\hat{R}_C$: METIS also leads to the most compact subregions. The $\hat{R}_C$ of our method is about 8% bigger than METIS, but 6% smaller than MADD, and about 2% smaller than direct CVT partitioning.

3) For *average separator angle deviation* $\hat{\delta}_\theta$: our algorithm has the smallest separator angle deviation. Our $\hat{\delta}_\theta$ is 40% smaller than METIS, 20% smaller than MADD, and about 5% smaller than direct CVT partitioning.

In Conclusion, our algorithm results in significantly smaller separator angle deviation than the METIS and MADD method, while preserving good workload balance and compactness. Compared with the direct CVT decomposition, our algorithm yields 40% smaller workload balance ratio, while have slightly better the compactness and the separator angle deviation small. This indicates that meshing based on our decomposition is about 20% faster than that using CVT decomposition. To achieve similar workload balance, the CVT method needs to use much more (in our experiments, more than 4 times) cells. Then as a side effect, this will lead to a significant increase in L_S and in singularities of the final quad mesh (see Section 5.3), which is undesirable.

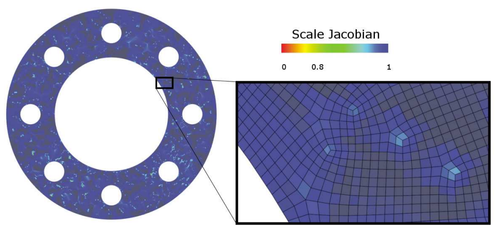
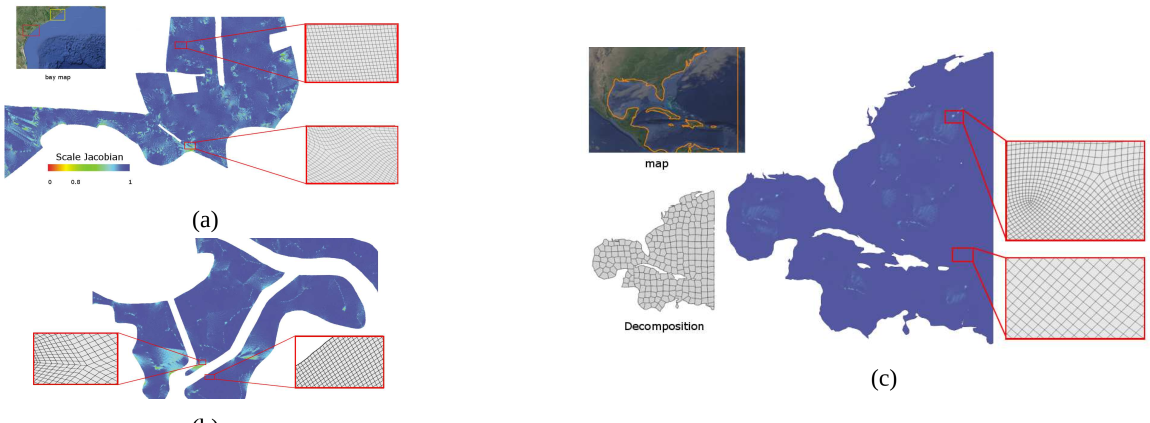
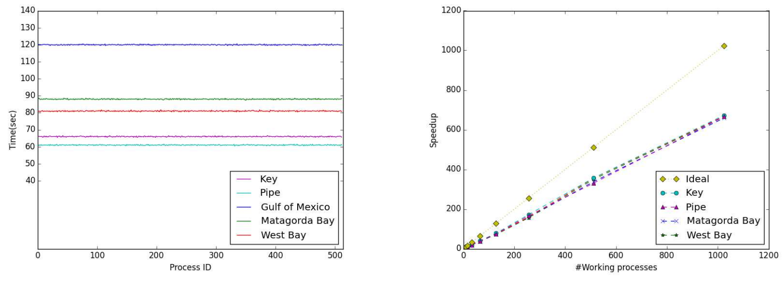


Fig. 7: Quad meshing result of the Pipe model, color-coded by elements' scaled Jacobian values. The color from blue to red indicates the mesh quality from high to low.



(b)

Fig. 8: (a, b, c) The quad meshes for West Bay (yellow region in bay map), Matagorda Bay (red region in bay map), and the Gulf of Mexico respectively. The quad meshes are color-coded in scaled Jacobian.



(a) (b)

Fig. 9: (a) The load balance for 512 working processes of the meshing of our dataset. (b) The parallel speedup of meshing: we test our algorithm using 4 to 1024 working processes. The yellow dot line is the ideal speedup, and the speed up of our algorithm on different models ranges from 2.78 to 601.5.

5.2. *Parallel Computational Efficiency*

Sub-regions are distributed to different working processors for simultaneous quad mesh generation using advancing front. We test the parallel computation efficiency on our datasets: the pipe model (Figure 7), the terrain near West Bay (Figure 8(a)), and the terrain near Matagorda Bay (Figure 8(b)) and the entire ocean region of the Gulf of Mexico (Figure 8(c)). Figure 9(a) shows the actual meshing time on each working processor. The working time on different processors are very balanced during the parallel execution.

Table 3: The Runtime Table for different partitioning algorithms. N_S is the number of subregions. The runtime is in minute, and includes the partitioning, meshing and relaxation time. Usually, the METIS is fastest, and our method has the second best efficiency.

Model	Key				Pipe				Matagorda				West Bay				Gulf of Mexico			
N_S	16	64	256	1024	16	64	256	1024	16	64	256	1024	16	64	256	1024	16	64	256	1024
L_∞-CVT	13.92	4.51	1.59	0.60	16.55	5.43	1.94	0.74	19.25	6.49	2.26	0.91	19.14	6.50	2.18	0.95	26.07	8.55	3.11	1.28
METIS	13.09	4.24	1.49	0.56	15.54	5.10	1.82	0.70	18.13	6.11	2.13	0.86	18.08	6.14	2.06	0.90	24.42	8.01	2.91	1.20
MADD	13.69	4.43	1.56	0.59	16.27	5.34	1.90	0.73	18.94	6.38	2.22	0.90	18.92	6.42	2.15	0.94	25.51	8.36	3.04	1.25
Our	13.60	4.40	1.55	0.58	16.07	5.28	1.88	0.72	18.78	6.33	2.20	0.89	18.70	6.35	2.13	0.93	25.27	8.29	3.01	1.24

We also run the experiments by changing the number of working processors. The total run time is shown in Table 3. METIS leads the fastest total running time, Our algorithm is about 7% slower than METIS, 1% and 3% faster than MADD and direct CVT respectively. Ideally, when the computation is evenly distributed to each processor, the speed up would be T/D, where T is the total time cost of our algorithm without parallelization, and D is the number of processors. But there are overheads of this divide-and-conquer pipeline, including the partitioning, data transmission, and post-processing. The workload balancing also affects the speed-up performance. Figure 9 (b) plots the speedup of our algorithm on different models. The yellow dot line is the ideal speedup, and our algorithm s' parallel speed up ranges from 2.78 to 601.5 when 4 to 1024 processors are used, respectively. The partitioning time usually takes up to 30% of the total time. To improve the meshing quality near the separator, we apply a Laplacian relaxation [35] near the separators after individually generated meshes are merged. Hence, a smaller L_S will reduce this post-processing time. In our experiments, the relaxation takes about 4% of the total computation time.

5.3. Meshing Quality Comparison

Model (N_S)	Sequential Algorithm	L_∞-CVT	METIS	MADD	Our Method
Key (32)	0.97 / 0.13 / 0.39 / 35	0.98 / 0.14 / 0.38 / 233	0.93 / 0.27 / 0.23/ 411	0.97 / 0.18 / 0.23 / 226	0.98 / 0.13 / 0.38 / 216
Pipe (16)	0.97/ 0.04 / 0.31 / 34	0.97 / 0.04 / 0.30 / 55	0.94 / 0.09 / 0.23 / 128	0.95 / 0.07 / 0.23 / 84	0.97 / 0.04 / 0.30 / 50
Matagorda Bay (512)	-	0.97 / 0.04 / 0.35 /325	0.93 / 0.16 / 0.21 / 413	0.96 / 0.08 / 0.27 /346	0.97 / 0.04 / 0.35 /305
West Bay (512)	-	0.96 / 0.05/ 0.36 / 235	0.93 / 0.12 / 0.24 / 562	0.95 / 0.07 / 0.28 / 321	0.98 / 0.05/ 0.36 / 225
Gulf of Mexico (1024)	-	0.97 / 0.04 / 0.36 / 3158	0.96 / 0.15 / 0.21 / 3491	0.96 / 0.09 / 0.23 / 3201	0.98 / 0.04 / 0.36 / 3104

Table 4: Mesh Quality Comparison between the sequential meshing algorithm, L_∞-CVT, METIS, MADD, and our algorithm. The sequential algorithm applies the advancing front without partitioning; and it only works on small models such as the Key and Pipe. N_S is the number of subregions. The four values: (1) average, (2) standard deviation, and (3) minimum of the scaled Jacobian, and (4) the number of singularities are listed to show meshing quality. The scaled Jacobian of our mesh is comparable to the sequential algorithm, but we have 20% more singularities. The average and minimal scaled Jacobians of our algorithm are 10% and 4% better than METIS and MADD, respectively. Our singularities are 40% and 20% fewer than METIS and MADD. Compared with L_∞-CVT, our algorithm leads to about 4% better average and minimal scaled Jacobians and 8% fewer singularities.

Table 4 compares the quality of final quad meshes generated by the sequential algorithm and four parallel algorithms using different partitioning methods. The sequential algorithm applies the advancing front to generate quad mesh without partitioning. For large model such as Matagorda Bay, West Bay and Gulf of Mexico the sequential algorithm failed to get a result. For each quad mesh, we compute four values, the (1) average, (2) standard deviation, and (3) minimum of scaled Jacobian, and (4) the number of singularities. Ideally, the scaled Jacobian should be 1. The scaled Jacobian of our mesh is comparable to the sequential algorithm, but we have 20% more singularities. Compared with other partitioning techniques, our average and minimal scaled Jacobians are 10% and 4% better than METIS and MADD respectively. Our singularities are 40% and 20% fewer than METIS and MADD. Compared with L_∞-CVT, our algorithm leads to about 4% better average and minimal scaled Jacobians and 8% fewer singularities. These experiments show that our algorithm produces higher-quality quad meshes than other partitioning algorithms.

6. Conclusions

We present a parallel quad mesh generation pipeline for large-scale 2D geometric regions. A main contribution of this work is the solving of data partitioning with effective incorporation of the geometric constraint on angles between separators. After partitioning, subregions are distributed to different processors for parallel mesh generation through advancing front. Finally, after composition, post-processing is performed near partitioning boundaries for refinement. We evaluate our partitioning and mesh generation algorithm in different experiments. Compared with other data partitioning stratifies, our new algorithm leads to better partition result and final meshing quality.

In the future, we will generalize this parallel pipeline for structured meshing of curved 2D manifolds and 3D solid regions. We will also investigate parallel meshing algorithms with controlled singularity numbers and distributions. For this purpose, singularity estimation needs to be incorporated in partition optimization; tessellation of subregions may be computed through parameterization-based mesh construction algorithms [36][37] which globally control the singularity distributions; and post-processing that allows the merging of nearby singularities may also be useful.

7. Acknowledgements

This research is supported by NSF IIS-1320959, IIS-1251095, and CNS-1205682.

References

[1] L. Linardakis, N. Chrisochoides, Delaunay decoupling method for parallel guaranteed quality planar mesh refinement, SIAM JSC (2006) 1394–1423.

[2] R. Löhner, A 2nd generation parallel advancing front grid generator, in: Proc. Int. Meshing Roundtable, 2013, pp. 457–474.

[3] T. Panitanarak, S. M. Shontz, Mdec: Metis-based domain decomposition for parallel 2d mesh generation, Procedia Computer Science 4 (2011) 302–311.

[4] D. Nave, N. Chrisochoides, L. P. Chew, Guaranteed: quality parallel delaunay refinement for restricted polyhedral domains, in: Proceedings of the eighteenth annual symposium on Computational geometry, 2002, pp. 135–144.

[5] Y. Ito, A. M. Shih, A. K. Erukala, B. K. Soni, A. Chernikov, N. P. Chrisochoides, K. Nakahashi, Parallel unstructured mesh generation by an advancing front method, Mathematics and Computers in Simulation 75 (5) (2007) 200–209.

[6] A. Shamir, A survey on mesh segmentation techniques, in: Computer graphics forum, Vol. 27, 2008, pp. 1539–1556.

[7] A. Agathos, I. Pratikakis, S. Perantonis, N. Sapidis, P. Azariadis, 3d mesh segmentation methodologies for cad applications, Computer-Aided Design 4 (6) (2007) 827–841.

[8] X. Chen, A. Golovinskiy, T. Funkhouser, A benchmark for 3D mesh segmentation, ACM TOG. 28 (3) (2009) 1–12.

[9] M. O. Freitas, P. A. Wawrzynek, J. B. Cavalcante-Neto, C. A. Vidal, L. F. Martha, A. R. Ingraffea, A distributed-memory parallel technique for two-dimensional mesh generation for arbitrary domains, Adv. Eng. Softw. 59 (2013) 38–52.

[10] J. Gould, D. Martineau, R. Fairey, Automated two-dimensional multi-block meshing using the medial object, in: Proc. of the 20th Int'l Meshing Roundtable, Springer, 2012, pp. 437–452.

[11] K. Brix, S. S. Melian, S. Müller, G. Schieffer, Parallelisation of multiscale-based grid adaptation using space-filling curves, in: ESAIM: Proceedings, Vol. 29, EDP Sciences, 2009, pp. 108–129.

[12] M. F. Mokbel, W. G. Aref, Irregularity in high-dimensional space-filling curves, Distributed and Parallel Databases 29 (3) (2011) 217–238.

[13] P. MacNeice, K. M. Olson, C. Mobarry, R. de Fainchtein, C. Packer, Paramesh: A parallel adaptive mesh refinement community toolkit, Computer physics communications 126 (3) (2000) 330–354.

[14] H. Ji, F.-S. Lien, E. Yee, A new adaptive mesh refinement data structure with an application to detonation, Journal of Computational Physics 229 (23) (2010) 8981–8993.

[15] A. Maximo, L. Velho, M. Siqueira, Adaptive multi-chart and multiresolution mesh representation, Computers & Graphics 38 (2014) 332–340.

[16] M. Attene, M. Mortara, M. Spagnuolo, B. Falcidieno, Hierarchical convex approximation of 3d shapes for fast region selection, in: CGF, Vol. 27, 2008, pp. 1323–1332.

[17] B. W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, Bell system technical journal 49 (2) (1970) 291–307.

[18] A. Pothen, H. D. Simon, K.-P. Liou, Partitioning sparse matrices with eigenvectors of graphs, SIAM Journal on Matrix Analysis and Applications 11 (3) (1990) 430–452.

[19] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM JSC 20 (1) (1998) 359–392.

[20] E. Vecharynski, Y. Saad, M. Sosonkina, Graph partitioning using matrix values for preconditioning symmetric positive definite systems, Journal on Sci. Computing 36 (1) (2014) A63–A87.

[21] H. Meyerhenke, Dynamic load balancing for parallel numerical simulations based on repartitioning with disturbed diffusion, in: Parallel and Distributed Systems (ICPADS), 2009 15th International Conference on, 2009, pp. 150–157.

[22] L. Linardakis, N. Chrisochoides, A static medial axis domain decomposition for 2d geometries, ACM TOMS 34 (1) (2005) 1–19.

[23] S. OWEN, M. STATEN, S. CANANN, S. SAIGAL, Q-morph: An indirect approach to advancing front quad meshing, Int. J. Numer. Meth. Engng 44 (1999) 1317–1340.

[24] X. Liang, M. S. Ebeida, Y. Zhang, Guaranteed-quality all-quadrilateral mesh generation with feature preservation, Computer Methods in Applied Mechanics and Engineering 199 (29) (2010) 2072–2083.

[25] S.-W. Chae, J.-H. Jeong, Unstructured surface meshing using operators, in: Proc. Int. Meshing Roundtable, 1997, pp. 281–291.

[26] T. D. Blacker, M. B. Stephenson, Paving: A new approach to automated quadrilateral mesh generation, International Journal for Numerical Methods in Engineering 32 (4) (1991) 811–847.

[27] D. Bommes, B. Levy, N. Pietroni, E. Puppo, C. Silva, M. Tarini, D. Zorin, Quad-mesh generation and processing: A survey, CGF 32 (6) (2013) 51–76.

[28] F. Kälberer, M. Nieser, K. Polthier, Quadcover-surface parameterization using branched coverings, in: CGF, Vol. 26, 2007, pp. 375–384.

[29] J.-F. Remacle, F. Henrotte, T. Carrier-Baudouin, E. Béchet, E. Marchandise, C. Geuzaine, T. Mouton, A frontal delaunay quad mesh generator using the l norm, International Journal for Numerical Methods in Engineering 94 (5) (2013) 494–512.

[30] M. Campen, L. Kobbelt, Quad layout embedding via aligned parameterization, in: CGF, Vol. 33, 2014, pp. 69–81.

[31] P. Bonami, L. T. Biegler, A. R. Conn, G. Cornuéjols, I. E. Grossmann, C. D. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya, et al., An algorithmic framework for convex mixed integer nonlinear programs, Discrete Optimization 5 (2) (2008) 186–204.

[32] B. Lévy, Y. Liu, Lp centroidal voronoi tessellation and its applications, ACM Trans. Graph. 29 (4) (2010) 119:1–119:11.

[33] Y. Liu, W. Wang, B. Lévy, F. Sun, D.-M. Yan, L. Lu, C. Yang, On centroidal voronoi tessellationłenergy smoothness and fast computation, ACM Trans. on Graphics 28 (4) (2009) 101.

[34] S. Fortune, A sweepline algorithm for voronoi diagrams, Algorithmica 2 (1-4) (1987) 153–174.

[35] W. Buell, B. Bush, Mesh generationła survey, Journal of Manufacturing Science and Engineering 95 (1) (1973) 332–338.

[36] S. Wan, Z. Yin, K. Zhang, H. Zhang, X. Li, A topology-preserving optimization algorithm for polycube mapping, Computers & Graphics 35 (3) (2011) 639–649.

[37] W. Yu, K. Zhang, S. Wan, X. Li, Optimizing polycube domain construction for hexahedral remeshing, CAD 46 (2014) 58–68.