All-Quad Meshing for Geographic Data via Templated Boundary Optimization

Lance Simons\textsuperscript{a}, Nina Amenta\textsuperscript{a,}\textsuperscript{*}

\textsuperscript{a}Department of Computer Science, University of California, Davis, One Shields Avenue, Davis, CA 95616, U.S.A.

Abstract
Motivated by the requirements of groundwater flow simulations in natural resource management, we give an algorithm that produces two-dimensional all-quad meshes of geographic regions with complex boundaries, often defined by rivers, watershed boundaries and geological features. Our algorithm fills the interior of a region with a regular grid, and fills the geometrically complex boundary regions using a global optimization over a choice of templated patches.

1. Introduction
Groundwater refers to water below the ground, in soil, sand and rock, as opposed to surface water such as rivers or lakes; it provides over half of our drinking water and is an important source for irrigation. Physical measurement of groundwater is expensive and time consuming, other than at specific points such as wells, so computer simulations are used extensively in groundwater management. Simulations are used to model groundwater storage, provide opportunities to compare different management schemes [1], and analyze the potential spread of contaminants [2]. In these simulations the physical characteristics of the material covered by a mesh element determine its water storage and flow characteristics. The goal is to determine flow into and out of each element, along with per-element recharge or discharge, based on incoming flow at sources (eg. along rivers) and outgoing flow at sinks (natural springs or man-made pumps).

Groundwater flow simulations solve diffusion equations using either finite difference or finite element analysis. While mixed quad-triangle meshes are most common for two-dimensional finite element solutions, several groundwater flow finite element codes can use, or even require (eg. the USGS’s SUTRA and CFEST solvers), quad meshes. Quad meshes also offer more accurate interpolation than triangular elements for a given element area [3], allow fewer elements for a given element diameter, and can improve simulation accuracy when oriented along the flow direction (which can often be estimated based on the underlying geology [4]). This work was done as part of a project

\textsuperscript{*} Corresponding author. Tel.: +1-530-754-5377; fax: +1-530-752-4767.
E-mail address: amenta@cs.ucdavis.edu

1877-7058 © 2017 The Authors. Published by Elsevier Ltd.
Peer-review under responsibility of the scientific committee of the 26th International Meshing Roundtable.
with the California Department of Water Resources to improve the repertoire of meshes available for their integrated GIS-to-FEA system [5].

Fig. 1: Overview of the meshing process, with an optional smoothing step. Colored using a perceptually-uniform color scale [6], with brighter colors indicating better internal angles.

The inputs to the meshing problem consist of the polygonal boundaries of the regions to be meshed, along with interior features such as rivers, material boundaries, political boundaries (e.g. water districts, counties, states), fault lines, and wells. We assume that the region boundary and internal features have been simplified to the scale desired for the mesh. These meshes may interlock with other meshes along common borders, so it is important that some perimeter segments are not subdivided. Our system creates size-uniform meshes at a fixed resolution, which is the first step in the workflow of our colleagues at the DWR (even when the option of graded meshes is available); regions of interest are then manually selected for further refinement. Aquifer properties (e.g. hydraulic conductivity, specific yield, specific storage, aquifer thickness) are supplied at every mesh element.

Since there are no mesh quality metrics that universally guarantee accurate simulation results [7], we have adopted an overall architecture that is largely agnostic to the choice of quality heuristics. We use a combination of quality heuristics, including maximizing the minimum angle and minimizing the maximum angle, quad aspect ratio defined as the ratio of minimum to maximum edge length, and minimization of the number of singularities. Singularities reduce the accuracy of linear FEM simulations, and increasing the order of the interpolation to compensate for them induces increased computational cost [8]; since we do not seek a graded mesh we can be aggressive about minimizing singularities.

Our method is illustrated in Figure 1. We place a section of a regular grid, with a user-defined orientation, in the interior of the region, leaving a channel between the grid and the region boundary. We then construct a collection of small patches, each of which might contribute to a covering of the channel, and score these by the quality of quadrangulation of the patch. We next select a covering of the channel by a subset of the patches. Finally, we smooth the resulting mesh.

The intuition behind this grid-based approach is that the difficulty of meshing a geographic region lies in the complexity of its boundary, to which the orientation of elements has to adapt; unlike the situation in which there are smooth boundaries along which elements should be aligned, here the boundaries force nearby element orientations to change frequently. A-priori gridding of the interior limits the singularities to this difficult region, and avoids unforced singularities in the interior.

Our approach produces all-quad meshes with element quality on par with modern tools that produce mixed meshes, while providing a large reduction in the number of singular points. We compare in depth in Section 4.
2. Prior Work

Triangulation software can produce nearly-uniform meshes which respect the input boundary edges as desired. These could be subdivided into quads using Catmull-Clark subdivision, at the cost of increasing the number of elements, or joined using triangle matching, as in the Blossom-Quad algorithm [9,10] or in the provable angle-bound algorithm of [11], or via more complex combinatorial operations, as in Q-Tran [12]. These methods produce a great many singularities, although advancements in topological improvement for quad meshes [8,13–15] can greatly reduce these while maintaining, or even improving, the mesh resolution and quality. Our method directly produces quad meshes with few singularities.

Much prior work in direct quad mesh generation is adapted to situations in which the boundary is the primary region of interest, with emphasis placed on generating high-quality perimeter elements at the expense of the interior. Paving [16], in particular, produces quad meshes in which a small number of singularities are concentrated in the interior, and regular element layers along the boundaries; it is implemented in CUBIT, and elsewhere. Q-Morph [17] uses combinatorial operations to transform a base triangle mesh to quads aligned with the boundary. Medial axis methods [18,19] minimize these interior singularities using the medial axis. For geological regions, however, the medial axis is generally as complex as the boundary itself. The circle-packing algorithm [20] produces bounded angles and a linear number of elements, but with arbitrary aspect ratios and singularities.

In recent years, interesting quad meshing algorithms have been developed in the geometric modeling community, using a base triangle mesh to solve a PDE for a function from which the mesh can be derived, with the boundary feature size significantly larger than the desired mesh size. Cross-field [21,22] methods solve for the mesh orientation (where a rotation by 90° is equivalent to a zero change in orientation), and then determine the mesh elements, leading to nearly square elements in most places. Spectral methods [23] and especially [24] compute a harmonic function that respects input features including the boundary, and then use the quadrilaterals formed by its Morse-Smale complex. So far neither of these approaches has been able to deal with complex geometric boundaries. On our inputs, the cross-field software of [22] approximates rather than matches the input boundary at mesh resolutions near the desired element size.

Our approach is a grid-based method. A grid-based method fills the interior of the region with a regular structure, and then connects the outer perimeter to this internal structure, typically using a template-based method.

In 3D, grid-based methods have proved to be the most successful approach to tetrahedral meshing of domains bounded by smooth surfaces; this problem differs from ours in that vertices can be placed arbitrarily on the smooth boundary. The internal structure in this case may be a lattice related to an octree, which provides internal grading [25,26], and they can provide excellent provable bounds on the resulting tetrahedral angles.

Similar approaches to quad-meshing in two dimensions use a grid as the internal structure, potentially with a quadtree subdivision if a graded mesh is desired [27,28]. Our approach is similar to these, differing mainly in how the channel is gridded. Again, this earlier work focused on piecewise-smooth boundaries that could be subdivided as part of the meshing process, by projecting internal vertices at the edge of the chosen grid subset onto the boundary. These projected points split the perimeter into new segments. Our method preserves the input boundary by spending more effort in stitching it to the interior grid. This results in uniform-size meshes that respect the input boundaries, matching the quality of the triangular and mixed meshes currently most popular for groundwater flow applications.

There are commercial software systems that produce meshes for groundwater flow, either as part of an integrated simulation system or as a pre-processor. The automatic mesh generator in GMS [29], the Groundwater Modeling System, uses triangle matching to create high-quality mixed triangle-quad meshes, and an all-quad paving implementation that is unable to honor internal boundaries within the mesh [29]. Argus ONE has a robust paving implementation that produces graded quad meshes that are finer than the boundary feature size near the boundary and can be coarser in the interior; the USGS MeshMaker package is based on this.

The open-source software package Gmsh, like GMS, builds a triangle mesh, then uses a matching step to produce mixed meshes, and optionally a Catmull-Clark subdivision step for all-quad meshes.
3. Construction

Like other grid-based approaches, our method fills the interior of the region with a regular grid. We remove elements from the inner grid that are near the perimeter, creating an unmeshed buffer zone that we call a channel. Channel width is a user-defined parameter that should be roughly one and a half times the desired element size. In order to preserve the geographic boundaries without introducing extra vertices, we use a wider channel and search for its best possible quadrilateral decomposition. We exploit the fact that the channel is narrow to perform a search of a great many (but not all) of the possible quadrangulations of the channel.

We organize this search by considering potential subdivisions of the channel into smaller polygons, with 4, 6 or 8 sides, that we call patches; we chose patches of this size as a trade-off between quality and computation time. We consider all possible local patches, select a quadrangulation of each one, and assign it a quality score, or rank. Then we select a compatible set of patches to produce the full channel covering that maximizes the minimum rank.

To assign a quadrilateralization to each patch, we consider a set of templates for the decomposition of 6-sided and 8-sided patches. Each template is applied to each patch, and quality of each of the resulting quads is measured; the quality of the template is the quality of its worst quad, and the rank of the patch is the quality of its best template.

The combinatorial search for the best covering of the channel by templates is the most algorithmically difficult part of the algorithm. We propose a process that simplifies the channel by iteratively merging extrusions and loops, by using a form of Dijkstra’s algorithm, eventually reducing the channel into a simple ring, which can be optimally traversed using the same algorithm.

We do not guarantee that our mesh is optimal. However, our algorithm is polynomial in the size of the mesh, and we demonstrate some high-quality results. The rest of this section covers the details of the algorithm and our implementation.

3.1. Input Format

The input region is given as a Planar Straight-Line Graph (PSLG). This is a collection of vertices, and an unordered list of segments that connect pairs of vertices, and which do not intersect except at vertices. We assume that input geometry has been discretized to approximately the desired edge length $h$, with all incoming input segments lengths in the range $\left[\frac{1}{2}h, 2h\right)$. Each segment must be preserved in the output mesh; generally, input segments may not be subdivided. This restriction may be relaxed in order to improve angle bounds, in which case perimeter segments may be split at most once.

We also need to require a density bound on the vertices, so that the neighborhood of radius $4h$ around any input vertex contains at most a constant number of other input vertices. Also, we note that small angles in the input force small angles in the output.

Some subset of the segments must form a fully closed outer boundary, defining the region to be meshed. The region may also contain holes, also defined by closed loops. The boundary of every region, including its holes, has to have an even number of edges; otherwise there is no decomposition into quadrilaterals.

We allow the PSLG to include interior features, which are paths that either terminate inside the region or divide the region into subregions. Both sides of any path that terminates in the interior are included in the boundary of the region, so they always add an even number of edges. Interior paths that split the region must also have an even number of edges. All of the input PSLG edges become outer channel edges.

In preprocessing, we combine the polygonal elements and identify the region and subregions to be meshed, and orient their boundaries counter-clockwise, by “walking” each connected component of the PSLG.

3.2. Grid

A regular grid with edge length $h$ is overlaid onto the region to be meshed. Any grid vertices outside the region are culled, as well as any vertices that are within a fixed distance $\delta$ from the perimeter of the region; we use $\delta = 1.4h$, but it can be set by the user.

Grid cells where all four corners remain are promoted to quads and form the interior of the mesh; remaining edges that connect grid vertices are enforced as edges that must remain in the output mesh, even if both of the two incident
grid cells have been removed; this can be seen in Figure 2. Isolated vertices with no neighbors are discarded. Grid edges that are not in the interior of the grid are called *inner channel edges*.

Just as the input segments were “walked” to produce polygons with consistent orientation, the edges of the interior regular grid are walked in the opposite direction, producing a polygon with inverted orientation. Specifically, we are walking along the half-edges that are adjacent to the channel, where some edges, such as the hanging edges on the right side of Figure 2, will be traversed in both directions.

### 3.3. Cross-Edge Construction

Our next goal is to identify cuts that split the channel into nicely-quadrilateralizable patches. In addition to the outer and inner channel edges, we create additional *cross edges* that divide the channel; cross edges will separate patches from other patches. Cross edges usually connect inner and outer channel vertices, but there are also usually cross edges that connect non-consecutive vertices along the outer perimeter, and similarly along the inner grid. In order to produce the family of candidate cross edges, we test all pairs of vertices within a multiple of the desired mesh edge length \( h \), keeping only those that do not cross an inner or outer edge. Our current test length is \( 4h \).

### 3.4. Patch Generation

We begin by finding all triangles in the graph formed by the inner and outer channel edges and the candidate cross edges. Then, any triangles that share a cross edge are glued together, forming a four-sided patch. This process of gluing along cross edges is continued between all patches (not just patches with the same number of edges), and all resulting patches with between 4 and 8 edges are kept as candidates for the following covering step. Note that we keep patches with odd edge lengths 5 and 7; as we will describe in Section 3.5, we sometimes introduce Steiner points along long cross edges, increasing a patch’s side number. Three iterations of joining are sufficient to create all possible patches of side-length at most eight.

There is significant overlap in the resulting patches; typically, a point in the channel is covered by hundreds of, and possibly a few thousand, overlapping patches, as seen in Figure 3.

### 3.5. Patch Subdivision and Ranking

Our next step is to quadrilateralize all patches, using a set of templates, and rank each patch by how well it can be divided into quads. The templates we use are shown in Figure 4, and their construction is discussed in Section 3.6. While the vertex positions along the perimeter of the patch are fixed, a template may have internal Steiner points which are free to move within the patch. For each template applied to each patch, we place the interior vertices by solving the Laplacian. The Laplacian does not optimize any specific mesh quality function; we use it for efficiency and because it produces generally high-quality results. We rank the results using the chosen quality metric. In our implementation, we use the maximum deviation of any quad angle from 90 degrees; we ignore quad aspect ratio at this point because all edges are guaranteed to be roughly the same size. We will use an explicit quality metric that includes aspect ratio when smoothing, as described in Section 3.10.

We use four possible templates for 6-sided patches and thirteen templates for 8-sided patches, but many of the templates are asymmetric and must be tested in different rotations on each patch. For any given patch, one rotation of one template will yield the highest quality subdivision, and that will determine the rank (and final quadrangulation) of that patch.
Cross edges may be split by adding a vertex at the midpoint, so that a 5-sided patch with a split edge will be meshed as a 6-sided patch, a 7-sided patch will be meshed as an 8-sided patch, and most 6-sided patches will be meshed as both 6-sided patches and 8-sided patches, and most 4-sided patches will be meshed as both 4-sided patches and 6-sided patches. In the patch selection processes described below, two patches that share a cross edge will only be considered neighboring if both patches agree on whether to split the shared cross edge or not. In practice, this means that a single patch will actually be \(2^{\text{cross edges}}\) different patches, each with a different set of split edges.

3.6. Template Generation

We consider templates that use at most three Steiner vertices to decompose 6-sided patches, and at most four for 8-sided patches. Some experiments using more Steiner vertices did not improve the results for any of our examples.

There are four possible 6-sided templates, one case for each of 0-3 Steiner vertices. There are other possible 2- and 3-vertex configurations, but they result in a degeneracy where two interior quads share two connecting edges. The first three templates cover all configurations listed in “Figure 10: Closure of boundaries with six nodes” in the seminal paper on paving [16]. Our fourth case improves the quad-decomposition of concave patches, which are not likely to arise in paving except perhaps on pathological input.

We used a recursive definition to enumerate all of the templates for 8-sided patches using at most four Steiner vertices. As shown in Figure 6, an 8-sided patch can be decomposed in one of five ways, corresponding to cutting off one, two, or three vertices from the patch. The leftmost template shows a single vertex being cut off; a single Steiner vertex must be added to ensure that the both subregions still have even perimeters; we do not consider adding three Steiner vertices in this case because it would produce a 10-sided patch. The two middle templates represent two vertices being cut off of the template, with zero and two Steiner vertices being added along the cut, respectively. The final templates show the patch being cut in half, with one or three vertices being added to ensure even perimeters. The resulting 6-sided and 8-sided regions are then recursively decomposed, stopping when at most four Steiner points are used. If the result is a quadrangulation of the original 8-sided patch, it is kept.
This recursive process produces the same topological decomposition many times. Eliminating duplicate topologies still leaves us with 1,006 unique 8-sided templates. The vast majority of these, however, are degenerate. Figure 5 shows two examples of degenerate templates, which are either doublets or non-manifold regions. Removing degenerate templates reduced the set of possible templates to only 36.

Even 36 templates, however, are expensive to try on each of hundreds of thousands of 8-sided patches, especially since each is tried in multiple rotations. To reduce the number of options, we tried the entire set on the Sacramento Valley and Merced datasets provided by the DWR, inserting the grid at different rotations and slightly different sizes to produce different channels, and then divided these into patches for testing. Interestingly, we observed that almost all templates were the best choice for at least one patch from this large dataset. But, templates that were used rarely were the best choices for “difficult” patches, which tended to be low-ranked, so that they were not selected for the final mesh anyway. Based on this experiment, we selected the thirteen most-used eight-sided templates, which are shown in Figure 4.

In Figure 4, the first of the six-sided templates, and two of the eight-sided templates, add no Steiner vertices. One might be tempted to think that these templates are redundant, since the quads they define should already have been found as four-sided patches. However, due to the fact that a cross edge may be split, the diagonals of these templates may not be cross edges, so we do need to consider them at this stage.

3.7. Megapatches

We refer to a patch that includes just one cross edge as a cap, a patch that includes two as a bead, and a patch that involve three or more cross edges as a junction. Any cover of the channel consists of some selection of caps and junctions, connected by sequences of beads.

A patch without any neighbors sharing one of its cross edges cannot be a part of a complete covering, and is removed from consideration. This may in turn result in further patches being removed.

Similarly, a cap represents a dead-end in the channel; see Figure 7. The cap can be removed by merging it with all of its neighbors. The resulting megapatches internally remember the originating patches, but can be treated like a single patch for the purposes of finding an optimal covering. If the resulting megapatch is again a cap, it can be further merged to form larger megapatches. This process of forming megapatches is repeated until no caps remain.

For some very thin input regions, where the boundary and features prevent an interior grid from being placed at all, the channel has genus zero. In this case, the best covering megapatch will represent the final meshing of the region, so our algorithm can terminate here.

In the more common case, a cap will produce a set of megapatches, each terminating at a junction patch, whose number of cross edges is reduced by one. The set of megapatches replaces the set of junction patches in the problem. Note that different junction patches might end up forming the same megapatch when they differ only in the cross-edge that ends up being merged into the megapatch. For every megapatch, we need to store only its highest-quality quad mesh, which determines the rank of the megapatch in further processing.

We can see this process as a graph algorithm, in which the patches are nodes connected by edges corresponding to their adjacencies along cross edges. Growing a cap into a collection of megapatches corresponds to growing a widest-path tree from the cap, terminating each branch when it encounters either another cap or a junction patch. Widest path is similar to shortest path, except that the width of a path is the minimum rank of any of its edges. Strictly speaking...
3.8. Covering the Channel with Patches

At this point, there are no caps remaining in the set of patches, the region has non-zero genus, and the covering we seek is a collection of cycles, possibly connected by junctions; Figure 1c shows the case where the cover is a single cycle of patches.

The case of a single cycle is solved using a variant of Dijkstra’s algorithm. An arbitrary perimeter edge along the cycle is selected; any single-cover of the region will contain exactly one of the patches incident to this edge. These patches are iteratively tested for the widest path around the loop back to itself, with the caveat that once a patch has been reached, the search forward from that patch must not traverse the cross edge that initially reached that patch. This condition is necessary to ensure a single-cover of the region. The widest path from all of the tested patches is taken to be the final cover for the region.

This approach can also be used to reduce the genus of a channel by one. The genus of a channel is determined by the number of connected components in the interior. Taking all patches incident to one of the isolated interior components yields a loop much like the genus-one case, except that patches in the loop may have cross edges that are not incident to the isolated component and thus do not have a neighbor across that edge in this subset of patches. Instead of taking only the widest path from the best patch, all paths from all patches are considered. For each combination of unsatisfied cross edges along the path, a new megapatch is formed, representing this particular path around the interior component. This can be seen in the lower portion of Figure 7. This will be performed for all isolated components in the interior except for the largest, which is left as the genus-one case.

3.9. Computational Complexity

We consider the complexity in terms of the outer channel edge count $p$ and the total area of the region. The interior grid produces $O(\text{area}/h^2)$ elements, which are easy to determine within that time bound. The final channel cover contains $O(p)$ quads, since each outer channel edge ends up incident to a single patch.

The complexity of our channel covering algorithm is greater than the output size, however, since we generate a significantly greater number of potential patches. Since we only connect vertex pairs within distance $4h$ with cross
edges, and we assume that there are not more than a constant number of these, a vertex will participate in at most a constant number of cross edges. A single vertex, in turn, will belong to only a constant (although large!) number of patches, giving us a total bound of $O(p)$ on the number of patches considered. For convenience, we will count each junction patch multiple times, in particular $k - 1$ times, where $k$ is the number of cross edges on the boundary of the junction patch; $k$ is at most eight.

Let us consider the process of forming a set of megapatches, starting from a cap. This is equivalent to forming a widest path tree, starting at the cap, with distinct junction patches at the leaves. The set of patches considered when computing this tree includes anything reachable from the cap, without passing through a junction. It can be computed in time $O(p \lg p_i)$ in the number of patches considered, using our current $O(n \lg n)$ implementation of widest path based on Dijkstra’s algorithm. There is an $O(n)$ algorithm [30] that we could also use here. The cap patch, and all bead patches examined while making the tree, will be removed in subsequent sub-problems and represented by one or more of the megapatches associated with the junction patches at the leaves. So each of them is processed once. Each (leaf) junction patch will end up with one of its cross edges inside the megapatch. The junction patch may appear in subsequent sub-problems, either as a bead patch, or as a junction with one fewer cross edges. Since a junction with $k$ cross edges loses a cross edge every time it is used, and then finally is counted once as a bead, it is counted $k - 1$ times.

Genus reduction is a relatively expensive process. There is a large (but constant) bound on the number of source patches considered, which we will call the patch density $d$. This means each genus reduction phase is a $O(dp \lg p)$ search, replacing the $p_i$ patches with $O(dj)$ megapatches for a loop with $j$ junctions.

Treating $d$ and $j$ as constant, the total time is $O(p \lg p)$ in the size $p$ of the input; using the theoretically better widest path implementation would bring this down to $O(p)$. Unfortunately, the constant is very large because the number of patches, while $O(p)$, is very large. Also, it is important to note that the solve is currently one of the fastest phases in the overall process, in practice, so we do not feel the need to replace it yet.

3.10. Smoothing

We perform smoothing as a post-processing step. Instead of the common Laplacian smoothing, we try to improve the angles directly, by repositioning each Steiner vertex such that it minimizes the deviation of all dependent angles from 90°. We currently use the Nelder-Mead Simplex algorithm [31,32], which is available in the mesh smoothing package Mesquite [33] and recommended for non-differentiable functions, such as taking the maximum of several metrics. We use the implementation in SciPy [34] for our angle-based smoothing. A benefit to this approach is that our ranking function may be directly used here; should we switch to a quality ranking function that is a combination of aspect ratio, maximum angle, and deviation from ideal size, that ranking function can still carry over to the smoothing step. An example of the smoothing is shown in Figure 8. We also employ a smoothing function that is a combination of angle deviation and stretch, the scaled ratio between the longest diagonal of the quad and the shortest edge, as seen in Figure 9.

An obvious benefit to the smoothing is that it relaxes the interior grid and allows elements along the perimeter to take on better shapes. However, the templates that were applied to the perimeter elements may no longer represent the ideal topological configuration. Local topological mesh improvement method may be able to improve these.
Angle histogram for the mesh above. The red bar, which includes all 90° angles, is drawn at 1/40 of actual height. The dashed lines show extreme angles.

Angle comparison between regular (blue) and smoothed (red) meshes. Note the spikes at 120° and 60°, corresponding to valence-3 and valence-5 singularities.

As expected, preserving element aspect ratios results in a drop in angle quality, but still improves the poorest angles.

**Fig. 9:** Analysis of angles in the resulting meshes, with optional smoothing steps. Smoothing only to enhance angles will produce arbitrarily skinny rectangular elements, creating visible “pinches” in the mesh.

### 4. Preliminary Results

Our working code is written in Python 2.7, and is executed with PyPy 5.1.0. Even though we take advantage of the Just-in-Time compilation of PyPy, the code is rather slow. Initially, Figure 9a, without smoothing, took just under 30 minutes to run. The smoothing to produce Figure 9c took an additional 10 minutes for 30 iterations; we require more smoothing passes, as it takes longer for the interior regions to converge to a configuration that is balanced with the perimeter.

We profiled the code to guide our optimization efforts. The most expensive phases of the algorithm, in decreasing order of overall runtime, are template application and patch ranking (89%), patch generation (5%), perimeter edge reordering and isolated region detection (3%), inner grid generation and channel formation (1%). The time taken to actually solve for the optimal channel cover was under four seconds, making it one quarter of a percent of the overall runtime; the most complex channel in this run was only genus two, so we expect quick solution times. A full 10% of overall execution time is spent simply measuring the quality of potential quadrilaterals within the patch ranking phase.

We expect that a speed-focused reimplementation in C++, taking advantage of ample opportunities for parallelism, would produce Figure 9a (again, without smoothing) in under a minute. However, our profiling has indicated that the template application phase was an obvious bottleneck. Parallelizing just that phase, by running 8 separate Python threads, reduced the total runtime to just 526 seconds. We further reduced the overall runtime to 165 seconds by porting just that code to C++, parallelized with OpenMP.
We cannot guarantee that any given channel can be meshed, but testing shows that finding a cover is very likely. We ran a series of tests against the Merced dataset, with different combinations of desired edge length $h$, channel width $\delta$, grid orientation and offset. The desired edge length was linearly sampled from the mean input edge length to one standard deviation below. Among 360 test cases, 18 failed to produce a covering of the channel. 8 of these succeeded after increasing the maximum cross edge length from $4h$ to $5h$, at the expense of almost doubled runtime and memory usage. 9 tests failed due to groups of internal edges that formed zero-area polygons, bypassing an orientation check and allowing them to be walked in the wrong direction. These cases succeeded after manually reversing the direction of the offending group of edges. There were only four grid offsets tested, with a zero or one-half phase shift in either dimension, and every failed test case had at least one corresponding successful test case that differed only in offset.
Table 1: Statistics for Merced County Meshes

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>-</td>
<td>10b,10g</td>
<td>10d</td>
<td>10c,10h</td>
<td>-</td>
<td>10i</td>
</tr>
<tr>
<td>Elements</td>
<td>3412</td>
<td>2161</td>
<td>2017</td>
<td>2017</td>
<td>2082</td>
<td>2082</td>
</tr>
<tr>
<td>Triangles</td>
<td>778</td>
<td>262</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Quads</td>
<td>2634</td>
<td>1899</td>
<td>2017</td>
<td>2017</td>
<td>2082</td>
<td>2082</td>
</tr>
<tr>
<td>Angle Min.</td>
<td>25.23</td>
<td>18.06</td>
<td>35.84</td>
<td>35.88</td>
<td>40.77</td>
<td>40.77</td>
</tr>
<tr>
<td>Max.</td>
<td>149.42</td>
<td>151.97</td>
<td>146.20</td>
<td>146.20</td>
<td>136.02</td>
<td>138.21</td>
</tr>
<tr>
<td>Singular Points</td>
<td>1146</td>
<td>286</td>
<td>134</td>
<td>134</td>
<td>153</td>
<td>153</td>
</tr>
<tr>
<td>Degree-3</td>
<td>277</td>
<td>55</td>
<td>93</td>
<td>93</td>
<td>101</td>
<td>101</td>
</tr>
<tr>
<td>Degree-5</td>
<td>849</td>
<td>228</td>
<td>41</td>
<td>41</td>
<td>52</td>
<td>52</td>
</tr>
<tr>
<td>Other</td>
<td>20</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Fig. 11: Replacing the inner grid with the boundary-aligned output of Instant Field-Aligned Meshes [22]. The target element count was set at one-quarter the desired amount, and then Catmull-Clark subdivision was applied to ensure an all-quad interior at the target resolution.

Figure 9 shows our method on the northernmost portion of the Sacramento Valley in California. While our objective function during the initial meshing process is based exclusively on element angles, that is not sufficient for the smoothing step, shown in Figure 9b, as it squishes some interior elements into arbitrarily-skinny rectangles. For the smoothing step, we use an objective function that takes element shape into account (Figure 9c).

We compare our method against Gmsh [35] in Figure 10, using data from Merced County, California, which was extracted from a mesh produced by GMS [29]. We show the meshes without color to better contrast the three methods. Angle histograms are shown for both methods; note that the histogram for Gmsh (Figure 10c) is skewed toward smaller angles as it also produces triangles. If the boundary conditions are relaxed, and perimeter edges are allowed to be split at most once, at their midpoint, we produce the mesh seen in Figure 10i; while the interior quality is quite similar, the poorest angles along the perimeter are significantly improved. Table 1 shows statistics for some of the meshes shown in Figure 10.

We are exploring the use of alternate interiors for our meshing process. For example, Figure 11 shows the output of Instant Field-Aligned Meshes [22] as a replacement for the interior grid. The result is a mesh where the interior aligns with the boundaries, while also strictly adhering to the complicated boundary of the region.

Acknowledgements

We would like to thank the California Department of Water Resources, for funding, data, and advice.
References