Quadcover for large deformation simulation

David Desobry

Université de Lorraine, Inria, F-54000 Nancy, France

Abstract

In this paper, we present a method for generating quadrilateral meshes that can withstand large deformations and remain valid throughout the simulation of hyperelastic materials such as rubber. Our approach involves starting with a CAD description of the 2D shape, triangulating the model, computing frame fields aligned on the boundaries, and applying a quadcover algorithm. When the simulation deforms the initial mesh to the point where one of its elements is inverted, our method uses the previous end state of the simulation to compute a new deformation-aware frame field and produce a deformation-aware quad mesh.

Keywords: Geometry processing, hyperelastic, remeshing, quadcover

Introduction

In recent decades, quadrilateral meshes have been widely used in deformation simulation software such as numea and others, as they tend to converge better than unstructured triangular meshes. These quad meshes have typically been generated using front-end advancement methods. However, more recently, algorithms based on frame fields have gained popularity due to their ability to produce high-quality quadrilateral meshes. In this paper, we will focus on the quadcover algorithm, which is particularly effective at generating quadrilateral meshes with a minimal number of singularity vertices. While frame field based algorithms have become increasingly robust, they have often been designed to generate meshes without any human intervention, limiting their flexibility. In contrast, we propose a method that allows for greater control over the number and position of singularities, enabling the generation of quadrilateral meshes that remain valid through large deformations.

Contributions. In this paper, we propose a modification to the quadcover algorithm for generating quadrilateral meshes suitable for deformation simulations using the Numea software. We describe the changes we made to the quadcover algorithm to make it compatible with computing quadrilateral meshes on the initial geometry of a CAD object, and then show that these improvements can go even further, making a quad mesh computed with quadcover valid for all geometries of a CAD model submitted to a deformation simulation.

The paper is structured as follows: Section 1 compares traditional simulation meshing methods to those used for computer graphics animations, to which our method is similar. The quadcover algorithm and its ability to create quadrilateral meshes are introduced in section 3. In section 4, we present implementation details that improve the quadcover algorithm's robustness when working on CAD databases that accurately represent what can be encountered in the world of numerical simulations. Section 5 describes how we can modify the method described in Section 4 to produce a quadrilateral mesh that is adaptable to multiple input geometries rather than just one. In addition, we present an iteration algorithm that alternates between quadrilateral mesh generation and simulation steps, allowing this quad meshing method to automatically adapt to the deformation simulation for which it was designed. Section 6 provides statistics on the impact of the quad mesh improvements presented in this paper on CAD models. The results of simulations performed with the Numea software on a quadrilateral mesh computed using the fully automatic process presented in this paper are then shown.

1. Related work

1.1. Typical quadrilateral mesh generation methods for numerical simulations

Traditionally, quadrilateral mesh generation for numerical simulation has been done using advancing front methods such as those proposed by Zhu [1], Owen [2], Lee [3], and Blacker [4], or by splitting or merging triangular meshes as proposed by Johnston [5] and Remacle [6]. These older meshing techniques are still commonly used due to their robustness and simplicity of implementation. However, it has been shown that poor quality quadrilateral meshes can significantly impact the convergence and accuracy of simulation results, particularly in the case of non-linear deformations [7]. To maintain good mesh quality despite deformations, previous work has employed remeshing techniques [1, 8], but these approaches can introduce slowness and imprecision to the results [8, 9]. In this paper, we aim to find a quadrilateral mesh that meets all quality requirements and can withstand deformations without the need for remeshing, using quad mesh generation techniques from the field of computer graphics.

1.2. Quadrilateral mesh generation for computer graphics animations

The problem of generating quadrilateral meshes that can withstand deformations during a numerical simulation is similar to the problem of generating quadrilateral meshes that can adapt to the steps of an animation sequence. In the field of computer graphics, several quadrilateral mesh generation techniques based on global parameterization have been developed, including QEX [10], Global Parameterization [11], an Approach for Quadrangulating Manifold Surfaces [12], Quadcover [13], Integer Grid Maps [14], Mixed Integer Quadrangulation [15], and Quantized Global Parameterization [16]. These methods aim to produce high-quality quadrilateral meshes from an input geometry, and

a survey of quadrilateral mesh generation techniques for computer graphics can be found in [17]. However, to the best of our knowledge, there is currently no flexible method based on the quadcover algorithm for generating quadrilateral meshes that remain of high quality despite deformations during a simulation. Recent work on quadrilateral mesh generation for animations has demonstrated that it is possible to adapt global parameterization techniques to generate high-quality quadrilateral meshes on multiple input geometries [18, 19], allowing for the minimization of worst-case quality among all geometries with high robustness. This motivates the idea that similar methods could be applied to numerical simulation, as high-quality and adaptable meshes are needed to compute high deformation simulations without the need for remeshing steps.

2. Constraints of simulation of high deformation on the input quad mesh

Non-linear simulations of high deformations refers to a computational method for modeling the behavior of objects that undergo significant changes in shape when they are subjected to external forces.

In other words, this method is used to predict how an object, such as a rubber seal, will deform when it is under stress or strain. The behavior of the object may not be linear, which means that the response to an external force is not proportional to the force applied. The method uses complex mathematical models to simulate the non-linear behavior of the object, allowing engineers to predict how it will behave in real-world scenarios.

2.1. Simulation method of our study case

In our case we want to study objects subject to highly nonlinear elasticity behaviors. To simulate these objects, it is recommended to use a material model that can capture large strains and nonlinearities, such as the hyperelastic model. Also, the numerical method should be able to handle the nonlinearities in the problem, we use a full Newton method.

Full Newton Method

Given the initial displacement: U_0

Iterate until convergence DN_L2 and EN satisfied:

1. Calculate the tangent stiffness matrix:
   $K(U_k)$
2. Solve the linear system for the change in displacement:
   $K(U_k)\Delta U = R(U_k)$
   where:
   $R(U_k) = F^{ext}(t) - F^{int}(U_k)$
3. Update the displacement:
   $U_{k+1} = U_k + \Delta U$

End iteration

Result: U_n is the converged solution

The convergence criteria DN_L2 and EN we use in our Full Newton Method are:

Displacement Convergence Criteria (DN_L2)

At the kth Newton iteration:

$\|\Delta U_k\|_{L2\text{-}average} \leq tol_U \cdot \|U_{cumul\_inc}\|_{L2\text{-}average}$

where:

$U_{cumul\_inc} = \Delta U_1 + \cdots + \Delta U_k$

ΔU_k is solution of $K(U_{total})\Delta U_k = R(U_{total})$

tol_U is selected tolerance (default 1e-3)

$\|.\|_{L2\text{-}average}$ is L2-norm averaged on the # of nodes

Energy Convergence Criteria (EN)

At the kth Newton iteration:

$|E_k| \leq tol_E \cdot |E_0|$

where:

$E_k = R(U_k) \cdot \Delta U_k$

E_0 = initial energy at current increment

tol_E = selected tolerance (default 1e-3)

2.2. Importance of a well conditioned stiffness matrix

The conditioning of a matrix refers to how sensitive the solution of a system of linear equations is to changes in the matrix or the right-hand side vector. A well-conditioned matrix leads to a stable and accurate solution, while a poorly conditioned matrix can lead to numerical instability and large errors.

Thus, having a well conditioned stiffness matrix is crucial in terms of convergence and accuracy of the simulation results. The more a quadrilateral mesh have elements with 90 degrees angles (ideally a rectangle) the more the stiffness matrix is well conditioned with a uniform distribution of eigenvalues. Our objective is that the quadrilaterals of our meshes are shaped liked rectangles in the zones where the external forces are the highest, to ensure numerical stability and a nice convergence rate of each Full Newton iteration.

2.3. Guidelines for the construction of a quadrilateral mesh adapted to this type of simulation

Here are some guidelines for constructing a better possible quadrilateral mesh for a simulation of high deformation with a Full Newton method and external forces applied on boundaries:

- Near-orthogonal angles: The quadrilateral elements should have angles close to 90 degrees to reduce the effect of shear deformation and minimize the presence of off-diagonal terms in the stiffness matrix. Elements with near-orthogonal angles have a better-conditioned stiffness matrix than elements with non-orthogonal angles.
- Boundary alignment: Alignment with the external force direction applied on the boundaries can improve the stability of the solution. If the mesh is not aligned with the deformation direction, the element edges can become distorted, leading to poor element quality and numerical instability.
- Local refinement: A good local refinement should be targeted at regions where the solution is expected to vary

rapidly or where high accuracy is required. This could be areas of high stress, material interfaces, or regions of high deformation. More importantly, it is crucial to have high quality elements (orthogonal and aligned with external forces) in these high-stress regions even if it means sacrificing the quality of the mesh elsewhere.

- Consider the computational cost: It is important to balance the need for accuracy with the computational cost when choosing the level of refinement in the mesh. A fine mesh can lead to a more accurate solution, but it also increases the computational cost of the simulation. The global sizing of the mesh should be done considering the time step size, the magnitude of the external forces and the convergence criteria of the simulation.

3. Quadcover for quadrilateral mesh generation

The quadcover algorithm is a method for generating quadrilateral meshes from a triangular mesh of a 2D domain. The first step in this process is to compute a frame field, which is a field of crosses aligned with the boundaries of the domain (see Fig.1). In addition to providing local orientations, the frame field also serves as a global segmentation of the domain, with the singularities of the field defining the irregular vertices of the resulting quadrilateral mesh (see Fig.2). From the frame field, we extract two vector fields (see Fig.3), which are then integrated to produce two scalar fields (see Fig.4). The integer isovalues of these scalar fields are then used to construct the quadrilateral mesh (see Fig. 5).

3.1. Frame field computation

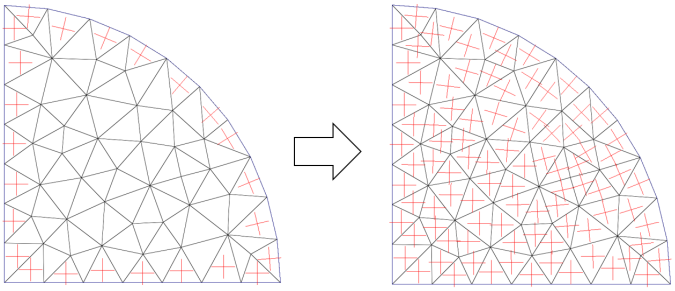

Figure 1: Computation of a frame field. Frames of boundary triangles are first fixed to be aligned with their boundary edge, then the frames of the inside triangles are obtained with a least square interpolation.

A 2D orthogonal frame can be represented as a cross, which is invariant under $\pi/2$ rotations. To capture this periodicity, Ray et al. [20] proposed representing the cross with the vector $(X, Y) = (\cos 4\theta, \sin 4\theta)$, where θ is the angle of the boundary edge. To obtain a smooth frame field, we first fix (X, Y) on the boundaries and then interpolate X and Y within the 2D domain (see Fig. 1). The optimization problem is to make the values of X and Y for neighboring triangles t and t' as similar as possible, while also satisfying the constrained values on the boundaries. Using the notation $\mathcal{N}(t)$ to represent the neighboring triangles of a triangle t, the optimization problem can be formulated as:

$$\arg\min_{X,Y} \sum_{t} \sum_{t' \in \mathcal{N}(t)} \left\| \begin{pmatrix} X_{t'} \\ Y_{t'} \end{pmatrix} - \begin{pmatrix} X_t \\ Y_t \end{pmatrix} \right\|^2 \quad (1)$$

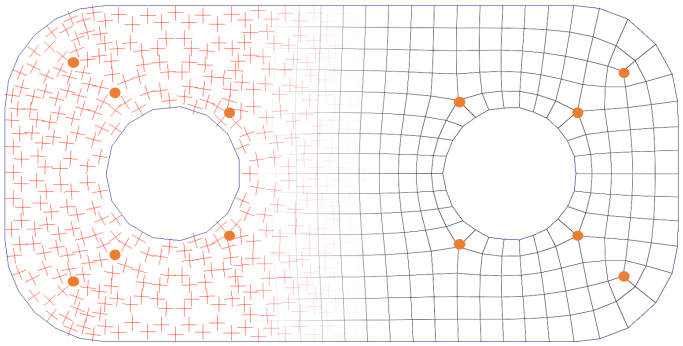

Figure 2: Singularities of a frame field are vertices of the output quad mesh that have a valence not equal to 4.

We use a least squares method to solve this optimization problem, just like we do for all other optimization problems in this article. From the computed X_t and Y_t values, we can calculate the angle θ_t of the cross field in all triangles of our input triangular mesh using the following formula: $\theta_t = \frac{1}{4}\tan^{-1}(\frac{Y}{X})$.

3.2. Integration with quadcover

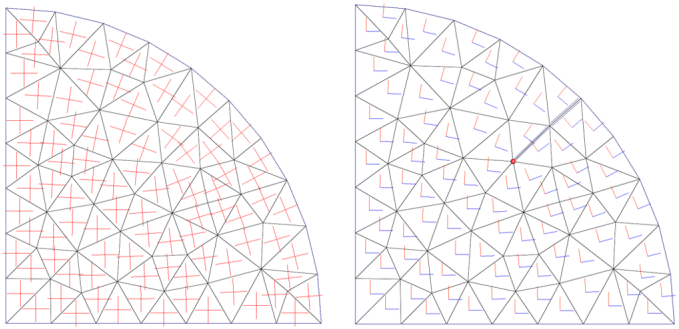

Figure 3: A frame field is brushed into two vector fields

The notation used in the following refers to the halfedge data structure notation of [21]. Specifically, if h is a halfedge that originates from the i_{th} corner of a triangle t: g_h corresponds to the (x, y) position of the i_{th} vertex of t, $h' = next(h)$ is the halfedge of t that follows h in the counter-clockwise direction, $h' = opp(h)$ is the halfedge from the adjacent triangle that is opposite to h and that shares the same vertices. $\mathcal{H}(t)$ is the set of the three halfedges of a triangle t.

The quadcover algorithm [13] provides an integer parametrization of a 2D domain from a frame field input. Given a triangular mesh $\mathcal{T}$ and two vectors per triangle a_t, b_t obtained from the frame field (as shown in Fig 3), we will compute two decimal values per triangle corner (represented by the halfedge that originates from this corner): u_h and v_h. These values u_h and v_h define two decimal functions u and v, which are linear per triangle, on the entire triangular mesh domain : let p be a point of a triangle $t \in \mathcal{T}$ with barycentric coordinates $\lambda_0, \lambda_1, \lambda_2$, we define the functions $u : p \mapsto \lambda_0 u_{h_0} + \lambda_1 u_{h_1} + \lambda_2 u_{h_2}$ and $v : p \mapsto \lambda_0 v_{h_0} + \lambda_1 v_{h_1} + \lambda_2 v_{h_2}$. These functions will be used to create our quadrilateral mesh: the points of our output quad mesh will be placed where $u(p)$ and $v(p)$ are integer values.

To create the quadrilateral mesh, we connect the points on the surface where either $u(p)$ or $v(p)$ is an integer. Since u and v are

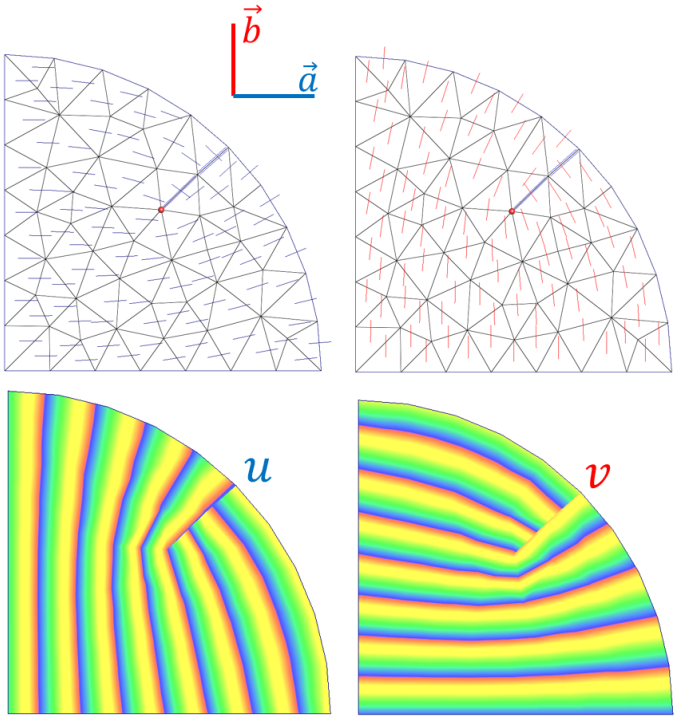

Figure 4: Each vector field a and b are integrated to produce respectively the scalar fields u and v.

linear per triangle, the integer lines in every triangle form segments that start and end on different edges. If the integer lines of u and v are continuous on all edges, meaning all adjacent triangles have integer lines that are aligned, then all output points are connected and form the desired quad mesh. The quadcover algorithm ensures the alignment of integer lines across all edges of the triangular mesh by enforcing two types of constraints, as described in the bellows paragraphs.

Seamless map constraint. :

A map defined on a 2D domain of a triangular mesh $\begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} u(x,y) \\ v(x,y) \end{pmatrix}$ is said to be seamless if the length of any halfedge $h \in \mathcal{H}(t)$ is the same that its opposite halfedge $opp(h) \in \mathcal{H}(t')$ for the u and v functions :

$$\begin{pmatrix} u_{next(h)} - u_h \\ v_{next(h)} - v_h \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}^{p_h} \begin{pmatrix} u_{opp(h)} - u_{next(opp(h))} \\ v_{opp(h)} - v_{next(opp(h))} \end{pmatrix}. \quad (2)$$

where $p_h \in \{0, 1, 2, 3\}$ are integer values determined after the frame field step, that minimize : $\left\| \begin{pmatrix} a'_t \\ b'_t \end{pmatrix} - \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}^{p_h} \cdot \begin{pmatrix} a_t \\ b_t \end{pmatrix} \right\|$.
In the figure 4, the vectors a_t are aligned with their neighbors a'_t except across two cut edges where it is aligned with $b_{t'}$, so the values $u_{next(h)} - u_h$ are constrained to be equal to $u_{opp(h)} - u_{next(opp(h))}$ everywhere except the two cut edges where it is constrained to be equal to $v_{opp(h)} - v_{next(opp(h))}$.

Integer map constraint. :
Having a seamless map does not suffice to compute a quadrilateral mesh. To do that we need a so called integer map, which is a seamless map where all triangle corners that share the same vertex have the same decimal part values for the u and v functions (but potentially different integer parts as we just want to align integer isolines).

For each halfedge h, we constrain integer alignment with the halfedge $next(opp(h))$ that starts from the same vertex:

$$\begin{pmatrix} u_{next(opp(h))} \\ v_{next(opp(h))} \end{pmatrix} - \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}^{p_h} \begin{pmatrix} u_h \\ v_h \end{pmatrix} = \begin{pmatrix} n_h \in \mathbb{N} \\ m_h \in \mathbb{N} \end{pmatrix} \quad (3)$$

Also, when a halfedge is a boundary, we want this halfedge to be an integer isovalue of either u or v, depending on if the boundary edge is normal to a_t or b_t, such that the output quad mesh shares the same boundary edges than the input triangle mesh : $u_h = u_{next(h)} \in \mathbb{N}$ or $v_h = v_{next(h)} \in \mathbb{N}$

Quadcover algorithm optimization. :
For each triangle $t \in \mathcal{T}$ and for each of its halfedges $h \in \mathcal{H}(t)$, we want to solve the following optimization problem with the constraints of equations 2 and 3:

$$\arg\min_{u,v} \sum_t \sum_{h \in \mathcal{H}(t)} \left\| \begin{pmatrix} u_{next(h)} - u_h \\ v_{next(h)} - v_h \end{pmatrix} - \begin{pmatrix} < a_t \ , \ g_{next(h)} - g_h > \\ < b_t \ , \ g_{next(h)} - g_h > \end{pmatrix} \right\|^2 \quad (4)$$

3.3. Quad mesh extraction

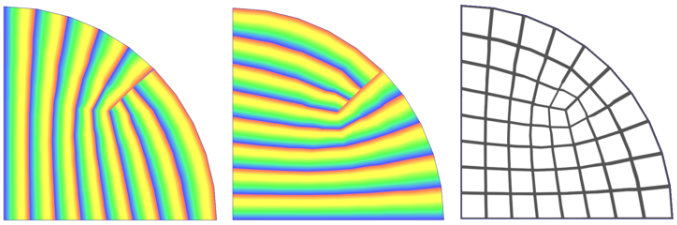

Figure 5: Extracting integer isolines of two quantized scalar fields permits to produce a quadrilateral mesh

The quadcover optimization of previous section permits to obtain two scalar functions u and v such that in each triangle the gradients of u and v are as close possible to the frame field vectors a_t and b_t. As u and v respects constraints of equations 2 and 3, the integer isolines of these functions are forming a quadrilateral mesh. The goal of this section is to accomplish the quadrilateral mesh extraction. This is done in a three-step process.

First, we want to locate the intersections between the integer isolines of u and v and the edges of our input triangular mesh $\mathcal{T}$. For each halfedge h, if there is an integer value between u_h and $u_{next(h)}$, we place an interpolation point on the edge at the integer location. We can then plot the integer isolines of u by drawing a line between each pair of intersection points in each triangle (if the two points have the same integer value for the u function). We do the same with the v function.

Second, if a triangle t has an integer line for both u and v functions and these integer lines intersect, we can place a vertex at the intersection, which belongs to the output quadrilateral mesh.

Third, now that we have placed all the vertices of our quadrilateral mesh, we can connect one vertex to its neighbors by recursively following the directions of the integer isolines through the neighboring triangles, until we reach another vertex.

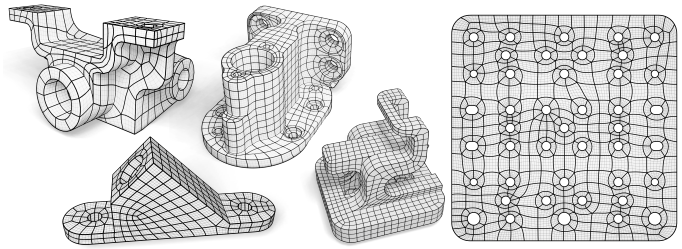

Figure 6: Examples of quadrilateral meshes on surfacic 2D or flat 2D domain, with feature line alignment constraints.

4. Implementation details for CAD models

Now that we have discussed the overall concept of the algorithm, we will present implementation details that improve the robustness of the method, specifically when dealing with CAD models that have feature lines that we want the output quad mesh to follow (as shown in Fig 6).

4.1. Frame field on low angle corners

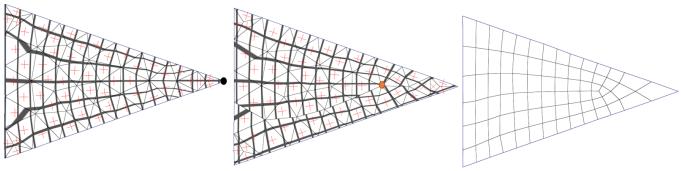

Figure 7: A classical framefield on a small angle corner suggest a valence 0 vertex (black dot, left). This frame field cannot be integrated while maintaining the boundary conditions. To solve this issue, we perform topological operations on the frame field so that the valence 0 vertex becomes a valence 1 vertex (at the cost of an interior vertex becoming a valence 3 vertex, represented by a red dot in the middle). The resulting frame field, with the modified topology, is integrable and can be used to create a valid quad mesh (shown on the right).

The first implementation of the pipeline described above was developed in the 2010s [17] and was able to generate high-quality quad meshes for most typical computer graphics models (as shown in Fig). However, when working with typical CAD models (as shown in Fig 6) where we want the output quadrilateral model to preserve sharp edges, a problem arises with low angle corners (as shown in Fig 7). Typically, a frame field will suggest that a boundary with angle α should be filled with $n = round(2\alpha/\pi)$ quadrilaterals. This means that for $\alpha < \pi/4$, we have $n = 0$, meaning that no quadrilateral is placed in the corner, resulting in a degenerate output (as shown in Fig 7). To prevent this from occurring, many options have been explored (such as in [22], desbruns, [23], directional, directional integ), using metric changes and/or non-orthogonality. However, in our case where we have multiple input geometries, we sometimes need to modify n even if it is not 0 (as discussed in section 5). We therefore need to use a more general resolution method that acts directly on the frame field topology. In section 5, we will also see that this algorithm can be used to modify vertex valences anywhere in the input model (not just on the boundaries).

As in [20], we refer to the frame field topology as the integer values p_h defined on the halfedge h of a facet t, opposite to the facet t'. We mentionned these p_h values in section 3.2 to express the constraints of the quadcover algorithm (eq 2 and 3). It contains information about how to compute the rotation between two frame field angles θ_t and $\theta_{t'}$. For example, the rotation between $\theta_t = -\pi + 10^{-5}$ and $\theta'_t = \pi - 10^{-5}$ should be $2 \cdot 10^{-5}$, rather than $\theta'_t - \theta_t = 2\pi - 2 \cdot 10^{-5}$, as the two crosses are nearly equal and the rotation between them should be close to 0. This type of problem did not occur with the $(X, Y) = (\cos 4\theta, \sin 4\theta)$ representation, which is $\pi/2$ periodic, but now we will switch to the more intuitive representation of a cross being represented by the angle of one of its 4 vectors: θ_t. From the θ_t angles obtained through the optimization in subsection 3.1, we define for all pairs of triangular facets sharing an edge: $p_h = round\left(\frac{\theta_{t'} - \theta_t}{\pi/2}\right)$.

This allows us to express the rotation between two adjacent crosses directly using their angles without encountering periodicity problems : $\theta_{t'} - \theta_t + p_h \cdot \pi/2$.

To have a smooth frame field (with as little rotation as possible between neighboring crosses), we can optimize the following problem according to the fixed topology given by the p_h values (h being the halfedge of t opposite to t'):

$$\arg\min_{\theta} \sum_{t} \sum_{t' \in \mathcal{N}(t)} \| \theta_{t'} - \theta_t + p_h \pi/2 \|^2 . \tag{5}$$

Performing this new optimization of the θ_t angles is not very useful, as we will end up with nearly the same θ_t values as before. However, the interesting part is that we can modify the p_h values before reoptimizing the θ_t values. This will result in a frame field with a new topology and different singularity locations (Figures 7, 8, 9).

For two adjacent facets t and t', we can increase the valence of the starting vertex of the halfedge h by 1 by increasing p_h and decreasing $p_{opp(h)}$ (while keeping $p_h = -p_{opp(h)}$ true). As a side effect, the valence of the destination vertex of h will be decreased by 1.

To solve the problem of a valence of $n = 0$ on a boundary vertex, we can take a halfedge starting from this vertex, and modify p_h and $p_{opp(h)}$ such that the valence becomes 1 (if the boundary vertex is only adjacent to one facet, we can split the triangular facet into three triangular facets with a new vertex at the center, so that the boundary vertex becomes adjacent to two facets). In this case, the destination vertex of the halfedge h will become a valence 3 singularity (if its starting valence was 4). This is an acceptable result because we will be able to obtain a valid parametrization from this frame field. However, in terms of the quality of the result, it is often much better to push the singularity further inside the domain. To do this, we can take a sequence of halfedges between the problematic boundary vertex and an inside vertex that will become a valence 3 singularity, and modify the p values on all of these halfedges. All intermediary vertices will receive an increase and a decrease in their valences, resulting in no valence change, so only the starting vertex and ending vertex will have a changed valence. Figure 8 shows that choosing a destination vertex that is a valence 5 vertex make it becomes a regular valence 4 vertex and improve the structure of the result quad mesh.

4.2. Implementation details on integration with quadcover

The integration step is the most prone to robustness issues. In subsection 3.2, we presented the optimization problem, and now we will focus on how to solve it and obtain scalar fields u and v such that for each triangle and its three halfedges h_0, h_1, h_2 : $\left\{\begin{pmatrix} u_{h_0} \\ v_{h_0} \end{pmatrix}, \begin{pmatrix} u_{h_1} \\ v_{h_1} \end{pmatrix}, \begin{pmatrix} u_{h_2} \\ v_{h_2} \end{pmatrix}\right\}$ expressed in the parametric domain (u, v) has a strictly positive area: $\det\left(\begin{pmatrix} u_{h_1} \\ v_{h_1} \end{pmatrix} - \begin{pmatrix} u_{h_0} \\ v_{h_0} \end{pmatrix}, \begin{pmatrix} u_{h_2} \\ v_{h_2} \end{pmatrix} - \begin{pmatrix} u_{h_0} \\ v_{h_0} \end{pmatrix}\right) > 0$ Having only positively oriented triangles in a (u, v) parametric

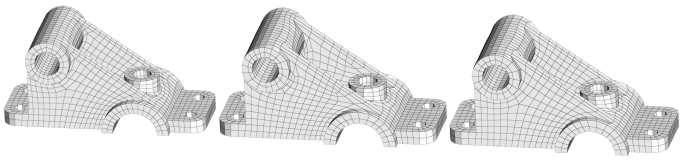

Figure 8: There are various ways to transform a valence 0 corner into a valence 1 corner. One approach is to add a valence 3 vertex near the corner (left), or farther away from the corner (middle). Another option is to remove a valence 5 vertex (right). Decreasing a valence 5 in order to increase a valence 0 corner provides the better structured result.

domain that respect the constraints of equations 2 and 3 guarantees that we will be able to obtain a valid quadrilateral mesh.

The first difficulty of the integration step comes from the constraints of equation 3, which add some integer constraints on decimal values (for example, the difference $u_{next(opp(h))} - u_h$ must be an integer if p_h is 0), making the optimization problem being a mixed integer programming problem. At first, we used to solve this problem directly with a mixed integer solver [15]. Later, we found a simpler and more robust solution [14]. The idea is to split the problem into two parts: first, compute u and v without integer constraints, and then use a quantization algorithm that takes this non-integer (u, v) map as input and outputs integer maps with integer alignment across edges as described in equation 3. The quantization algorithm that we are using is [**?**]. It simplifies the input triangle mesh until only integer variables are remaining (singularities and boundary vertices), which reduces the problem to an Integer Linear Programming (ILP) problem. Instead of solving it with an ILP solver, which can take a lot of time, it uses a greedy heuristic approach to solve it. While there is no guarantee on the optimality of the found integer values, this approach is faster compared to ILP resolutions and typically provides valid integer solutions. From the integer values on the simplified mesh, we can interpolate decimal values on all the non-integer vertices of our input triangular mesh and obtain two scalar fields u and v such that all triangles have a strictly positive area in the (u, v) parametric domain, with integer alignments across each edge.

The quantization algorithm takes as input a parametric domain represented by (u, v) in which all triangles have a positive orientation and conform to the seamless constraints outlined in equation 2. It produces a parametric domain that meets both the seamless and integer constraints outlined in equations 2 and 3. However, the optimization method outlined in equation 4 ensures seamlessness but does not guarantee positive orientation of triangles. We will present a new iterative process that will allow us to converge towards a parametric domain represented by (u, v) that satisfies the seamless constraints and has positively oriented triangles.

Even without integer constraints, ensuring that all triangles in the input mesh have a positive area in the (u, v) parametric domain from an input frame field topology can be challenging, and in some cases may even be infeasible (see subsection 4.3). After producing an integrable frame field topology (for example with the method described in 4.3), the integration remains a tricky part because when we solve the optimization problem of equation 4, we do not have a condition on the positivity of the triangle areas in the (u, v) parametric domain. However, as the vectors a_t and b_t of the frame field are positively oriented, the more the gradients of u and v get closer to a and b, the more the triangles will be positively oriented in the (u, v) domain. We introduce (μ, υ) a parametric domain that have for gradient the vectors a_t and b_t : let h_0, h_1, h_2 being three halfedges in couterclockwise order of triangle t, we assign the values

$$\begin{pmatrix} \mu_{h_0} & \mu_{h_1} & \mu_{h_2} \\ \upsilon_{h_0} & \upsilon_{h_1} & \upsilon_{h_2} \end{pmatrix} = \begin{pmatrix} 0 & < a_t\ ,\ g_{h_1} - g_{h_0} > & < a_t\ ,\ g_{h_2} - g_{h_1} > \\ 0 & < b_t\ ,\ g_{h_1} - g_{h_0} > & < b_t\ ,\ g_{h_2} - g_{h_1} > \end{pmatrix} \tag{6}$$

This (μ, υ) is positively oriented by construction, but as no reason to respect the seamless constraints, as opposed to (u, v). When the optimization 4 provides (u, v) triangles negatively oriented, our method is to do another optimization with a bigger weight on the optimization lines that correspond to these negative triangles. This will make these negative area (u, v) triangles closer to the (μ, υ) triangles (increasing the chances that they become positive oriented). As a side effect, the (u, v) triangles that already have positive area will move away from the model given by the (μ, υ) triangles, potentially becoming negative oriented. However, even if this happens, the next iteration will increase their coefficient lines to make them positive again.

This iteration process, although not as reliable as other methods (e.g. foldover free maps [**?**]), is sufficient for CAD model datasets (see section 6) and can easily be applied to handling positive orientation on multiple input geometries, as opposed to the previous work.

4.3. Resolving non-integrable frame field topologies

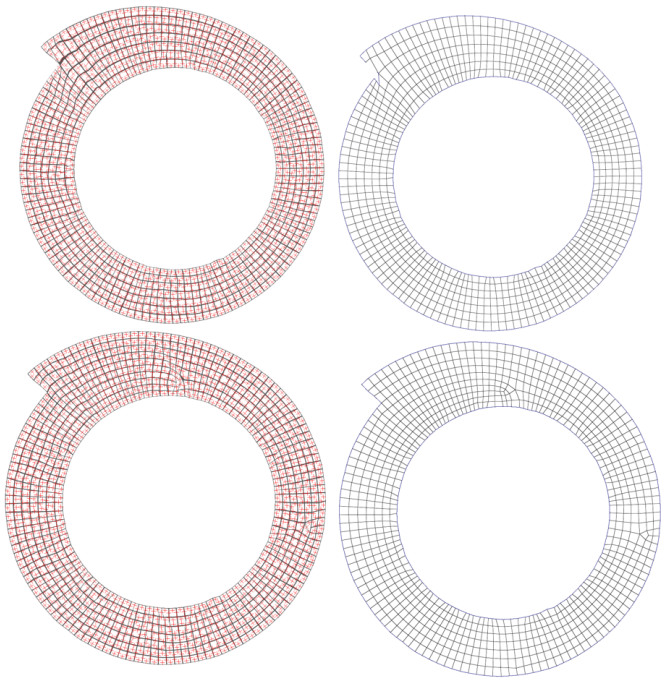

Figure 9: It happens that a perfectly smoothed frame field cannot be integrated, due to boundary alignment conditions that cannot be satisfied during the integration process. The issue can be resolved by doing some random modifications on the framefield topology. Here, two random edges have received a framefield topology modification, resulting in the creation of two dipoles of valence 3 and valence 5 vertices.

When the integration process does not produce triangles with positive area in the parametric domain (u, v) everywhere, it is likely that a solution does not exist with the given frame field topology (as shown in Figure 9). To address this issue, we can alter the frame field topology in order to obtain one that allows the construction of a valid (u, v) domain.

To modify the topology of the frame field, we use the same method described in Section 4.1, but with a starting vertex that is

located on the interior of the triangular mesh rather than at a low angle. This results in a decrease in valence for the starting vertex and an increase in valence for the destination vertex. While it is preferable to select the starting and destination vertices to produce the highest quality quad mesh possible, this problem is NP-complete and there is no known heuristic to find even a good solution.

Therefore, in our automatic method, we randomly select two neighboring facets (t and t') of the input triangular mesh and modify the frame field topology by increasing p_h and decreasing $p_{opp(h)}$. If the starting and destination vertices of the halfedge h were not previously singular, they will become valence 5 and valence 3 vertices, respectively, in the output quad mesh after this modification. The inclusion of this singularity dipole allows for greater flexibility in finding a valid (u, v) domain. If this does not produce a successful outcome, we can repeat the process with different facets. In general, only a few additional singularity dipoles are necessary to obtain a (u, v) parametric domain where all triangles have positive orientation, using the integration algorithm described in Section 4.2 on the new frame field topology.

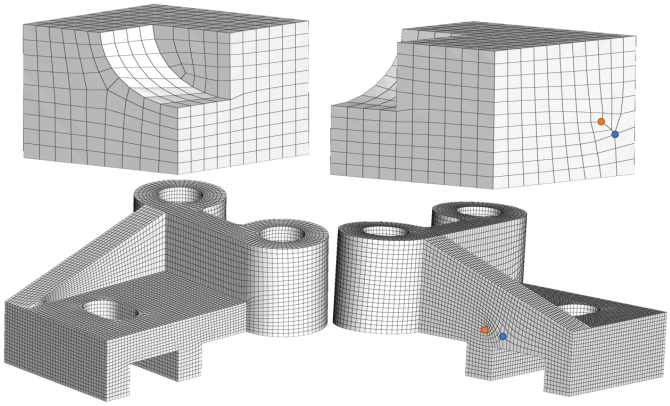

Figure 10: In our experiments on the mambo dataset (described in subsection 6.1), we found that adding a single randomly placed dipole of singularity (valence 3 vertex in red, valence 5 vertex in blue) was sufficient to solve the models that had non-integrability problems.

5. Multiple geometries in input

The previous section describes a pipeline that takes a triangular mesh as input and produces a high-quality quadrilateral mesh with the same boundaries and feature edges. During a deformation simulation, the positions of the points in the input mesh change over time, providing information about the nature of the deformation that the quad mesh must handle. If the initial quad mesh is not able to handle the full deformation simulation because quadrilaterals become degenerate, we can modify the initial quad mesh to ensure it remains valid when the same deformations are applied.

In the following sections, we will consider multiple geometries as input. For example, if we use a deformation simulation to generate these geometries, we will have the positions of each point in the input triangular mesh for all steps of the simulation.

5.1. Iterative improvement of a quad mesh

Our goal is to perform a numerical simulation of large deformation with a software using quadrilateral meshes. To do that, we need to generate a quadrilateral mesh that will remain of high quality when deformed by the simulation. To achieve this, we will repeatedly alternate between generating a quadrilateral mesh and running a simulation with it. The more accurately the quadrilateral mesh fits the deformations, the more precise the simulation results will be. On the other hand, the more accurate the simulation is, the more it will provide input geometries that can be used to create quadrilateral meshes that better fit the deformations. Precisely, if a simulation does not produce the desired results when using a particular quadrilateral mesh, we can divide the mesh into a triangular mesh and use the positions of its vertices at each simulation step to recalculate a quadrilateral mesh that will maintain a high level of quality with the deformations deduced from these input point positions.

This multi-step quadrilateral meshing algorithm is as follows:

$\mathcal{T} \leftarrow$ triangular mesh with initial geometry
$Q \leftarrow$ quad mesh from quadcover algorithm on $\mathcal{T}$
while The numerical simulation applied on Q fails **do**
  $\mathcal{T} \leftarrow Q$ splited into a triangular mesh
  $Gs \leftarrow$ point positions of $\mathcal{T}$ for all simulation steps achieved
  $Q \leftarrow$ polygeometries quad meshing algorithm on T, Gs.
end while

5.2. Topological problems of a frame field

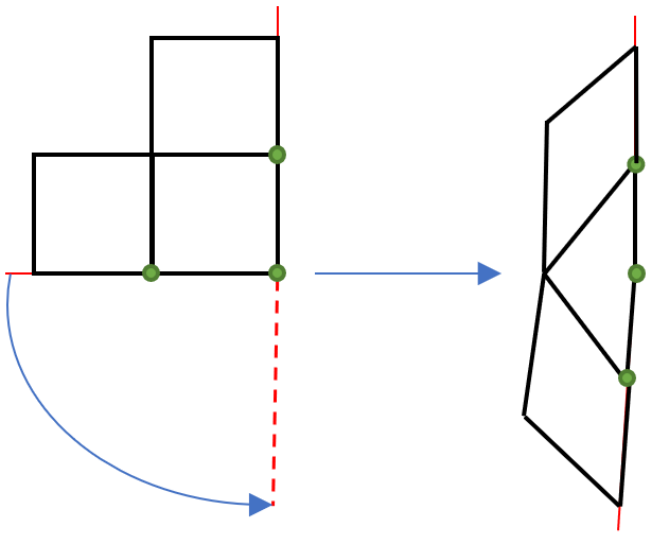

Figure 11: After the deformation of the boundary, the number of quadrilaterals adjacent to the boundary vertex become incompatible with computing a valid quadrilateral mesh.

In Subsection 4.1, we mentioned a topological error that can occur with a frame field when dealing with low-angle corners. The solution was to avoid valence $n = 0$ on all boundary edges. When dealing with multiple geometries as input, valence $n = 1$ or $n = 2$ can also lead to degenerate quadrilaterals when the boundary points are moving (see Figure 11). We do not encounter this problem with a single geometry as input because a frame field always places $n = \lfloor 2\alpha/\pi \rfloor$ when facing a α-angle corner due to the boundary alignment constraints. With multiple geometries, we have multiple boundary alignment constraints that we may not be able to satisfy simultaneously. In this case, we apply the boundary constraints of the initial geometry and optimize the frame field as described in Subsection 3.1. If we detect that on the k_{th} geometry of our inputs, the angle α_k of a boundary angle suggests a valence greater than the one suggested by the initial geometry:

$\lfloor 2\alpha_k/\pi \rfloor > \lfloor 2\alpha_0/\pi \rfloor$, we perform a topological change (as in Subsection 4.1) such that the valence becomes $n = \lfloor 2\alpha_k/\pi \rfloor$.

As a result, we have a new frame field topology that takes into account the highest boundary angles among all the geometries to ensure we have enough degrees of freedom to deform the boundary edges when starting a new simulation with a better-fitted quad mesh (see Figure 12 and 13).

5.3. Frame field model for each geometry

The final step is to smooth the frame field (using Equation 5) for each geometry k with the new topology p_h and the boundary constraints given by the point positions of geometry k, denoted as g^k_h. This results in multiple parametric domains (μ^k, υ^k) for each geometry k of the input, given by the frame field of the geometry:

$$\begin{pmatrix} \mu^k_{h_0} & \mu^k_{h_1} & \mu^k_{h_2} \\ \upsilon^k_{h_0} & \upsilon^k_{h_1} & \upsilon^k_{h_2} \end{pmatrix} = \begin{pmatrix} 0 & < a^k_t\ ,\ g^k_{h_1} - g^k_{h_0} > & < a^k_t\ ,\ g^k_{h_2} - g^k_{h_1} > \\ 0 & < b^k_t\ ,\ g^k_{h_1} - g^k_{h_0} > & < b^k_t\ ,\ g^k_{h_2} - g^k_{h_1} > \end{pmatrix} \tag{7}$$

With a valid frame field topology and multiple models of the (μ^k, υ^k) parametric domains (one per input geometry), we can start the integration step, which aims to be valid according to all the (μ^k, υ^k) models.

5.4. Polygeometries quad meshing algorithm

As a reminder, the (μ^k, υ^k) parametric domains given by the frame fields are not directly usable for computing quadrilaterals because they do not satisfy the constraints in Equations 3 and 2. We will use them to construct a model (μ, υ) that can be used as input to the algorithm in Subsection 4.2 to compute a parametric domain (u, v) that satisfies these constraints.

Our approach is to initialize (μ, υ) to the mean of the (μ^k, υ^k) values at all corners of the input triangular mesh. The goal is to ensure that all triangles in the input mesh have positive area in the (μ, υ) domain for all geometries. In rare cases, this may not be directly the case, so we can try to weight the geometries that are encountering problems more heavily in the mean sum. We never faced a case where it was not possible to have all triangles with positive orientation in the (μ, υ) domain for all geometries, but it could happend and be solved by increasing the quality of the input triangulation.

Now we assume that all triangles are positive oriented in the (μ, υ) domain for all geometries, we can follow the same integration process as in Subsection 4.2: we compute (u, v) to be as close as possible to (μ, υ) while respecting the integration constraints. If some triangles have negative area in the (u, v) domain for some geometries, we increase the weights of these triangles in the least square optimization to make (u, v) closer to (μ, υ). All triangles will end up being positive oriented in the (u, v) domain for all geometries.

At the end of these steps, we have a (u, v) parametric domain that we can use as input to the quantization algorithm described in subsection 4.2 before using the quad meshing algorithm described in subsection 3.3 to obtain a quad mesh that is adapted to the multiple geometries of our deformation simulation.

6. Results

In this paper we (1) presented a complete pipeline to generate a quadrilateral mesh from a triangular mesh, with tips to improve the robustness of the algorithms of the state of the art without too much increasing the implementation complexity (section 3 and 4). Then we (2) presented other modifications to the state of the art algorithms to compute a quad mesh that remain of high quality despite deformations deduced from a triangular mesh with vertices that have evolving positions in time (section 5).

6.1. Robustness test of (1)

Our implementation of (1) is a C++ program that consists of 1000 lines of code and does not use any external libraries. We conducted experiments on two datasets: a flat 2D dataset containing approximately 1000 models (cited in reference [flat2Dds]) and a 3D surfacic dataset containing approximately 150 models (cited in reference [mambo]). In all cases, we were able to successfully generate a valid quad mesh in less than a minute.

To demonstrate the significance of the improvements described in section 4, we conducted additional experiments on the mambo dataset where we excluded one or more of these improvements. When we omitted all of the improvements, the success rate was 30%. Excluding only improvement 4.1 resulted in a success rate of 50%. Omitting only improvement 4.2 resulted in a success rate of 40%. Finally, excluding only improvement 4.3 resulted in a success rate of 95%.

Based on these results, we can conclude that our quad meshes have singularities corresponding to the frame field topology of improvement 4.1 in 95% of cases. This means that the quadrilateral meshes we generate have a low number of irregular vertices (with valence $\neq 4$). In the remaining 5% of cases where the frame field topology of improvement 4.1 is not sufficient to compute a valid (u, v) parametric domain, improvement 4.3 modifies the frame field topology to a valid one. The addition of less than 3 randomly placed dipoles of singularities suffices to pass the 5% remaining cases. Even in these cases, the resulting quadrilateral meshes still have a low number of irregular vertices and are of high quality (see Figure 10).

6.2. Test of (2) with the solver Numea

To evaluate the effectiveness of generating a quad mesh that is optimized for deformation simulation, we conducted tests using Numea, a software specialized in solving nonlinear elasticity simulations, which are known to be prone to convergence problems. We obtained 5 rubber seal simulations from Hutchinson, a company that performs these simulations for their customers. These simulations involve simulating the deformation of flat 2D sections of rubber seals.

Normally, it takes Hutchinson an average of 3 weeks to generate a quadrilateral mesh that can withstand the simulation. For two of these 5 challenging tests (see Fig 14 and 15), we were able to complete the deformation simulation using the quadrilateral mesh generated with sections 3 and 4 on the initial geometry of the 2D domain. For two other tests (see Fig 16 and 17), we were able to complete the deformation with fewer than 3 steps of alternating between quad meshing with section 5 and trying a simulation with the newly created mesh.

For the last simulation, we were unable to complete the deformation simulation without a human intervention. Indeed, this simulation required tweeking of simulation parameters to be completed due to convergence issues that were not related to a low quality element in the input mesh. In this case, our meshing method provided no advantage over other methods.

7. Conclusion

In this paper, we present an improved version of the quadcover algorithm, which is more robust when applied to CAD models.

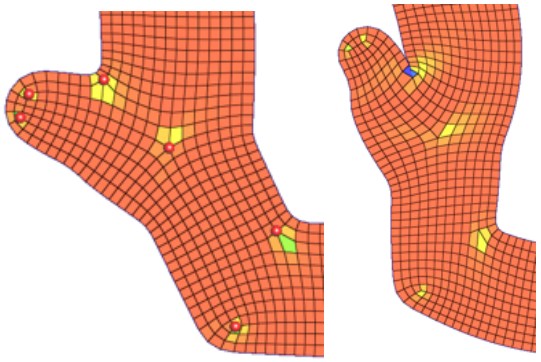

Figure 12: Optimal singularity positions for an initial geometry can lead to poor quality quadrilaterals after a deformation.

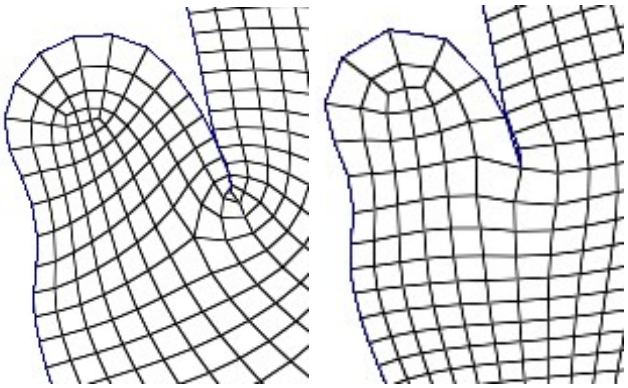

Figure 13: Using all the geometries from the previous iteration of the failed simulation automatically places 4 adjacent quadrilaterals at the boundary point where the deformation is most pronounced.

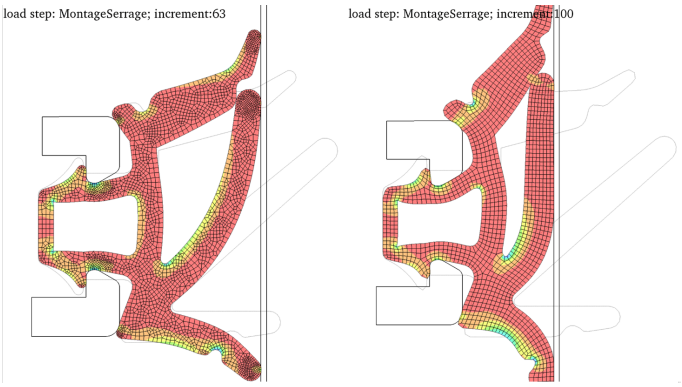

Figure 14: The method outlined in this paper for generating quad meshes automatically produced a high-quality mesh that sustained the 100 iterations of a high deformation simulation (as shown on the right). In contrast, a mesh generated using a Delaunay triangle merging method failed to converge at iteration 63.

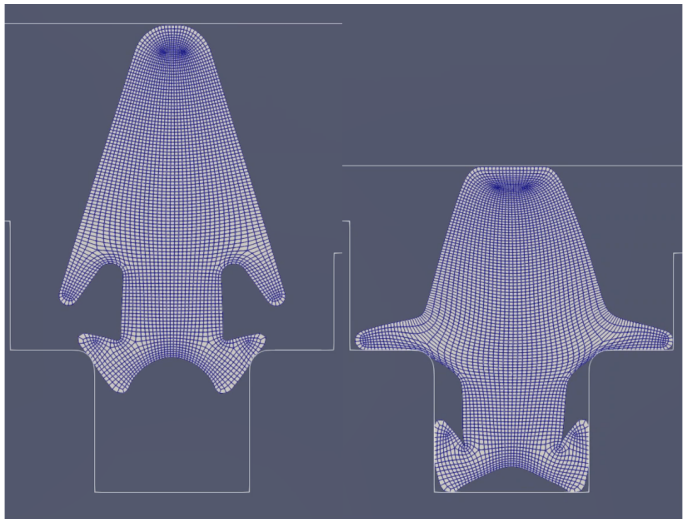

Figure 15: The quadrilateral meshes we propose have a boundary alignment that makes them inherently resistant to compression simulations.

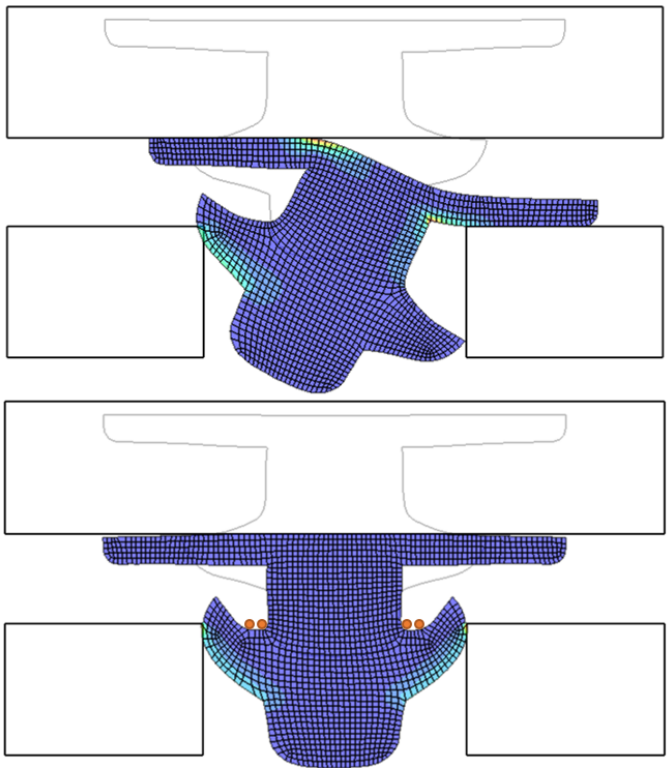

Figure 16: When the same deformation simulation is run on two different quad meshes, the results can be very different. Due to convergence issues, the seal in the first case has broken through its holder. In the second case, the seal has bent to fit normally in its holder thanks to the automatic addition of boundary valence 3 vertices on the red dots.

We have found that this algorithm has significant time-saving potential for industrial use in generating quadrilateral meshes for deformation simulation. However, like other automatic quadrilateral meshing methods, it may sometimes require manual adjustments when used for nonlinear simulation. One advantage of our frame field method is that the initial quadrilateral mesh has a simple combinatorial structure with few singularities, which makes it easier to make local modifications, either during the meshing process by changing the frame field topology (we can displace singularities this way), or after the meshing process on the output quad mesh. When simulating a 2D domain with multiple materials, we can treat the material boundaries as feature edges and have the edges of the generated quadrilateral mesh automatically follow these edges (see Fig 18). However, our method does not offer an automatic way to apply different sizing to different regions of the same model, which is a frequent request in the case of simulation. To do this, manual intervention is required, either by placing dipoles of singularities at desired locations during the meshing process (as shown in Figure 19) or by directly modifying the output quadrilateral mesh, as described

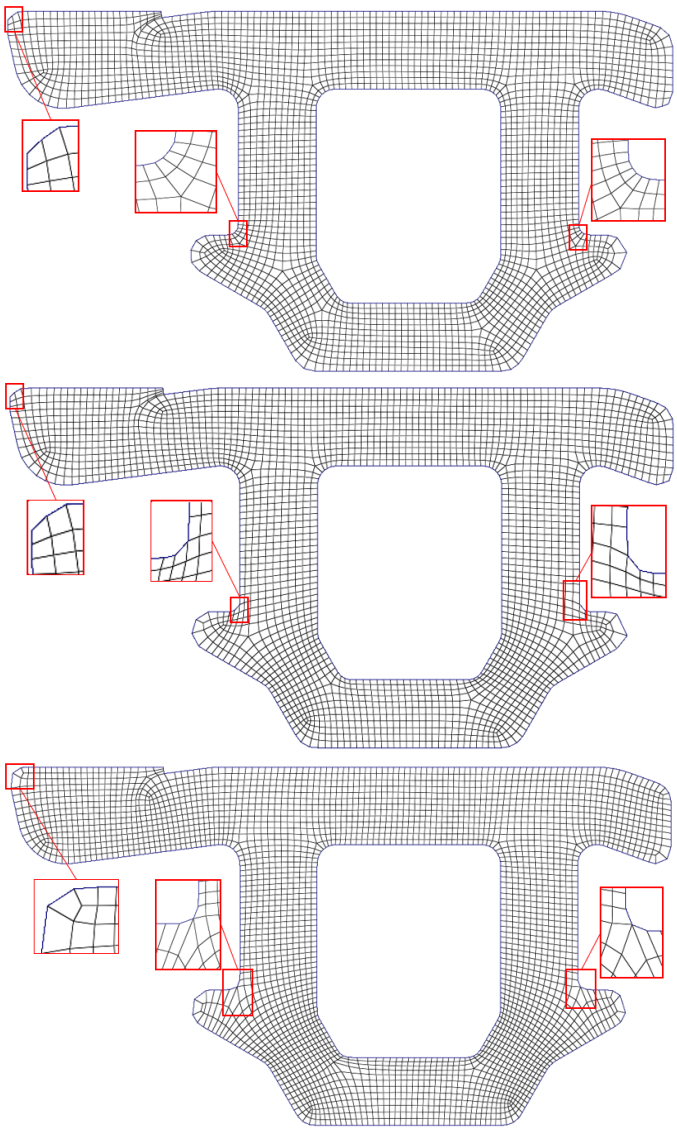

Figure 17: Top: a quadrilateral mesh generated from the initial 2D domain. Middle: quadrilateral mesh derived from all geometries of the first failed simulation, Bottom: quadrilateral mesh derived from all geometries of the failed second simulation. The bottom quadrilateral mesh passed the simulation due to better overall geometry and more adapted boundary valences than the previous meshes (red squares).

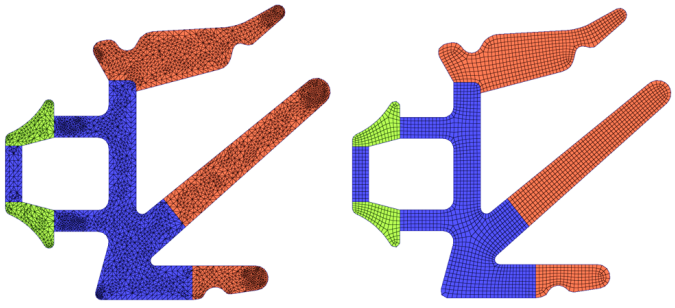

Figure 18: The method presented in this paper generates quadrilateral meshes that align naturally with material frontiers by treating them as boundaries.

in [24].

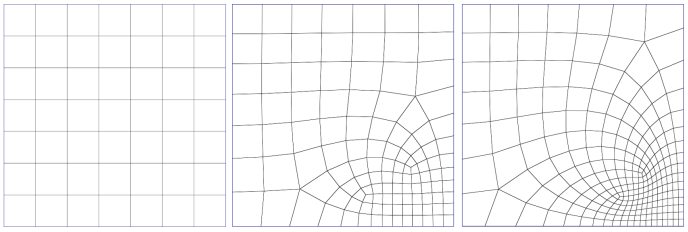

Figure 19: Local refinement thanks to the placement of two dipoles of singularities (two vertices of valence 3 and two vertices of valence 5).

References

[1] J. Zhu, M. Gotoh, Automatic remeshing of 2d quadrilateral elements and its application to continuous deformation simulation: part i. remeshing algorithm, Journal of Materials Processing Technology 87 (1-3) (1999) 165–178.

[2] S. J. Owen, M. L. Staten, S. A. Canann, S. Saigal, Q-morph: an indirect approach to advancing front quad meshing, International journal for numerical methods in engineering 44 (9) (1999) 1317–1340.

[3] C. Lee, S. Lo, A new scheme for the generation of a graded quadrilateral mesh, Computers & structures 52 (5) (1994) 847–857.

[4] 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.

[5] B. P. Johnston, J. M. Sullivan Jr, A. Kwasnik, Automatic conversion of triangular finite element meshes to quadrilateral elements, International Journal for Numerical Methods in Engineering 31 (1) (1991) 67–84.

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

[7] D. Arnold, D. Boffi, R. Falk, Approximation by quadrilateral finite elements, Mathematics of computation 71 (239) (2002) 909–922.

[8] N.-S. Lee, K.-J. Bathe, Error indicators and adaptive remeshing in large deformation finite element analysis, Finite Elements in Analysis and Design 16 (2) (1994) 99–139.

[9] A. Srikanth, N. Zabaras, An updated lagrangian finite element sensitivity analysis of large deformations using quadrilateral elements, International Journal for Numerical Methods in Engineering 52 (10) (2001) 1131–1163.

[10] H.-C. Ebke, D. Bommes, M. Campen, L. Kobbelt, Qex: Robust quad mesh extraction, ACM Transactions on Graphics (TOG) 32 (6) (2013) 1–10.

[11] D. Bommes, T. Lempfer, L. Kobbelt, Global structure optimization of quadrilateral meshes, in: Computer Graphics Forum, Vol. 30, Wiley Online Library, 2011, pp. 375–384.

[12] R. Viertel, B. Osting, An approach to quad meshing based on harmonic cross-valued maps and the ginzburg–landau theory, SIAM Journal on Scientific Computing 41 (1) (2019) A452–A479.

[13] F. Kälberer, M. Nieser, K. Polthier, Quadcover-surface parameterization using branched coverings, in: Computer graphics forum, Vol. 26, Wiley Online Library, 2007, pp. 375–384.

[14] D. Bommes, M. Campen, H.-C. Ebke, P. Alliez, L. Kobbelt, Integer-grid maps for reliable quad meshing, ACM Transactions on Graphics (TOG) 32 (4) (2013) 1–12.

[15] D. Bommes, H. Zimmer, L. Kobbelt, Mixed-integer quadrangulation, ACM Transactions On Graphics (TOG) 28 (3) (2009) 1–10.

[16] M. Campen, D. Bommes, L. Kobbelt, Quantized global parametrization, Acm Transactions On Graphics (TOG) 34 (6) (2015) 1–12.

[17] D. Bommes, B. Lévy, N. Pietroni, E. Puppo, C. Silva, M. Tarini, D. Zorin, Quad-mesh generation and processing: A survey, in: Computer Graphics Forum, Vol. 32, Wiley Online Library, 2013, pp. 51–76.

[18] G. Marcias, N. Pietroni, D. Panozzo, E. Puppo, O. Sorkine-Hornung, Animation-aware quadrangulation, in: Computer Graphics Forum, Vol. 32, Wiley Online Library, 2013, pp. 167–175.

[19] J. Zhou, M. Campen, D. Zorin, C. Tu, C. T. Silva, Quadrangulation of non-rigid objects using deformation metrics, Computer Aided Geometric Design 62 (2018) 3–15.

[20] N. Ray, B. Vallet, W. C. Li, B. Lévy, N-symmetry direction field design, ACM Transactions on Graphics (TOG) 27 (2) (2008) 1–13.

[21] D. Muller, F. Preparata, Finding the intersection of two convex polyhedra, Theoretical Computer Science 7 (2) (1978) 217–236.

doi:https://doi.org/10.1016/0304-3975(78)90051-8.
URL https://www.sciencedirect.com/science/article/pii/0304397578900518

[22] D. Desobry, Y. Coudert-Osmont, E. Corman, N. Ray, D. Sokolov, Designing 2d and 3d non-orthogonal frame fields, Computer-Aided Design 139 (2021) 103081.

[23] D. Desobry, F. Protais, N. Ray, E. Corman, D. Sokolov, Frame fields for cad models, in: International Symposium on Visual Computing, Springer, 2021, pp. 421–434.

[24] M. Lyon, D. Bommes, L. Kobbelt, Cost minimizing local anisotropic quad mesh refinement, in: Computer graphics forum, Vol. 39, Wiley Online Library, 2020, pp. 163–172.