Proceedings of 2001 DAC
27th Design Automation Conference
September 9-12, 2001 Pittsburgh, Pennsylvania, U.S.A.

DETC2001/DAC-21149

QUAD-LAYER: LAYERED QUADRILATERAL MESHING OF NARROW TWO-DIMENSIONAL DOMAIN BY BUBBLE PACKING AND CHORDAL AXIS TRANSFORMATION

Soji Yamakawa
soji@andrew.cmu.edu
The Department of Mechanical Engineering
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

Kenji Shimada
shimada@cmu.edu
The Department of Mechanical Engineering
Carnegie Mellon University
Scaife Hall 318
5000 Forbes Avenue
Pittsburgh, PA 15213

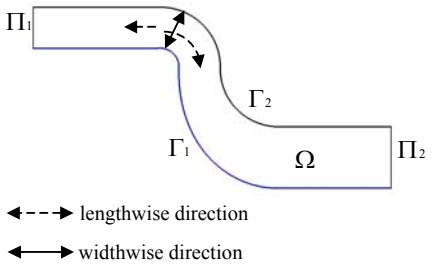
ABSTRACT

This paper presents a new computational method for quadrilateral meshing of a thin, or narrow, two-dimensional domain. An output mesh of our method is well-shaped and either single-layered, multi-layered, or partially multi-layered. Element sizes can be uniform or graded. A high quality, layered quadrilateral mesh is often required for finite element analyses of a narrow two-dimensional domain with a large deformation such as the analysis of rubber deformation or sheet metal forming. Our method consists of two steps: (1) extraction of the skeleton of a given domain by the discrete chordal axis transformation, and (2) discretization of the chordal axis into a set of line segments and conversion of each of the line segments to a single quadrilateral element or multiple layers of quadrilateral elements. In both steps we use a physically-based computational method called bubble packing to discretize a curve into a set of line segments of specified sizes. Experiments show that the accuracy of a large-deformation FEM analysis can be significantly improved by using a well-shaped quadrilateral mesh created by the proposed method.

1 Introduction

This paper describes a new computational method that creates a high-quality quadrilateral mesh of a thin, or narrow, two-dimensional domain such as the cross-section of a tire, a thin metal part, or a formed metal joint. Our method can arbitrarily control the aspect ratios of the quadrilateral elements, and the mesh created by our method is appropriate for the finite element analysis involved with large deformation of a narrow cross-section.

Our method deals with a narrow two-dimensional domain as shown in Figure 1 (a), and outputs a single-layered quadrilateral mesh as shown in Figure 1 (b), a multi-layered quadrilateral mesh as shown in Figure 1 (c), or a partially multi-layered mesh as shown in Figure 1 (d). In the rest of this paper, we use the term 'length' to refer to the element size in the lengthwise direction shown in Figure 1 (a), and the term 'width' for the element size in the widthwise direction in Figure 1 (a).



(a) input domain

(b) single-layered quad mesh | (c) multi-layered quad mesh | (d) partially multi-layered quad mesh

Figure 1 An example of an input narrow two-dimensional domain and three types of output layered quadrilateral meshes

High quality quadrilateral mesh of this type of domain is often required in finite element analysis of various artifacts including both narrow objects and the thin cross-sections of flat surfaces. Some examples include:

- Extruded parts - In extrusion a round heated billet is placed in a chamber of a large press and forced through a die by a hydraulic ram, forming a long profile shape with a constant cross-section.
- Sheet metal parts - In continuous roll forming, metal sheet is fed through a series of top and bottom rollers, gradually bending and forming the sheet to a target profile. The attainable shapes are somewhat similar to extrusions except that the profile is limited to the original thickness of the sheet.
- Fastening of sheet metals - The formation of a formed metal joint involves a deformation of a narrow cross-section. In order to fasten two sheet metals, their edges are lined up and rolled together so that they enfold each other.
- Rubber parts - A tire typically supports heavy objects such as a car or truck, and the cross-section deforms greatly by the loading. A tire consists of many thin layers of different materials, including layers of rubber, and each of them has a narrow cross-section.
- Paint and coating - The analysis of the coating over a sheet metal part such as an automobile panel often needs a layered quadrilateral mesh. In order to tolerate fractures or cracks caused by heat, moisture or other external factors, a car body is covered by layers of coatings. Because some of the layers are very thin, they often have to be meshed into a layered mesh when a finite element analysis is performed.

Conventional automatic quadrilateral meshing methods, however, are not designed for a narrow domain, and they often generate skewed elements. Those approaches, including advancing front [1-4] and triangle pairing [5, 6] commonly distribute nodes uniformly along the boundary curves, which produce poorly-shaped elements in regions where the curvature is high or where the curvature changes rapidly. In order to avoid those ill-shaped elements, the user needs to adjust manually the boundary node distribution.

Our algorithm creates well-shaped, layered quadrilateral elements automatically over a narrow two-dimensional domain in two stages: (1) extraction of the skeleton of a given domain by discrete chordal axis transformation, and (2) discretization of the chordal axis into a set of line segments and conversion of each of the line segments to a single quadrilateral element or multiple layers of quadrilateral elements. In both stages we use a physically-based computational method, called bubble packing, to discretize a curve into a set of line segments of specified sizes

The rest of this paper is organized as follows. We describe the problem statement in Section 2. We discuss related research in Section 3, and we show an overview of our approach in Section 4. We then explain our key technique of one-dimensional meshing that we call 'one-dimensional bubble packing' in Section 5 followed by the extraction of the chordal axis and quadrilateral mesh generation in Sections 6 and 7. We show some results in Section 8 and conclude at the end.

2 Problem statement

Our method takes as input a narrow two-dimensional domain Ω and a desired element length distribution $d(\mathbf{x})$, where $\mathbf{x}$ is a two-dimensional coordinate, and outputs a quadrilateral mesh. The domain Ω is a two-dimensional region enclosed by four curves $\Gamma_1, \Gamma_2, \Pi_1$ and Π_2 as shown in Figure 1 (a). Γ_1 and Γ_2 are boundary curves in the lengthwise direction, and Π_1 and Π_2 in the widthwise direction. For a narrow domain Γ_1 and Γ_2 are much longer than Π_1 and Π_2. These boundary curves can be given as parametric curves or as polylines.

The output quadrilateral mesh is a single-layered mesh such as Figure 1 (b), a multi-layered mesh such as Figure 1 (c), or a partially multi-layered mesh such as Figure 1 (d). When the width of Ω is smaller than the user-specified width threshold everywhere in Ω, the output will be a single-layered mesh. In contrast, if the width of Ω is larger than the user-specified threshold in any part of Ω, the mesh will be either a multi-layered mesh or a partially multi-layered mesh. Details on the layers of an output mesh are described in Section 6.

The element length distribution function, $d(\mathbf{x})$, specifies the distribution of desired element lengths, or element sizes, in the lengthwise direction. There are several possible ways to define this function $d(\mathbf{x})$:

- Uniform : All elements should have the same length.
- Uniform aspect ratio : All elements should have the same aspect ratio (the ratio of the length and width).
- Curvature-based : Desired element lengths are given based on the curvature of the lengthwise boundary curves Γ_1 and Γ_2. Element sizes should be smaller in a high curvature region in order to minimize the geometric approximation error.
- Analysis-based : In adaptive remeshing, desired mesh lengths are defined based on a previous finite element analysis result. Smaller element lengths should be defined in a region where the solution changes dramatically.
- User-specified : The user specifies element sizes manually at selected points in the domain. The complete definition of $d(\mathbf{x})$ is then obtained by

interpolating the specified element lengths at these points.

3 Related work

There are several approaches proposed for quadrilateral meshing of a general two-dimensional domain, but they often generate a poor quality mesh when applied to a narrow domain.

Several variations of the advancing front algorithm exist, typically creating quadrilateral elements from the boundary toward the interior of a domain. Blacker and Stephanson propose an algorithm called 'Paving' to create a quadrilateral mesh for a two-dimensional domain [1]. Their method creates quadrilateral elements layer by layer. Cass and Benzley propose a generalized paving algorithm for a three-dimensional surface [2]. White presents a modified paving algorithm [7] that creates quadrilateral elements one by one rather than layer by layer. Owen et al. propose an advancing-front algorithm which converts a triangular mesh into a quadrilateral mesh [3]. Thompson and Soni propose an advancing-front algorithm that creates a quad-dominant or hex-dominant mesh [4].

Another type of approach to quadrilateral meshing is node placement and connection. Shimada et al. propose a method that creates a quad-dominant mesh by packing square cells in the domain [5]. Itoh et al. present an algorithm that converts a triangular mesh into a quadrilateral mesh by pairing triangles [6]. Their method first creates ideal node locations for the quadrilateral mesh, and then creates a Delaunay triangulation based on the node locations, and finally converts the triangular mesh into a quadrilateral mesh. Viswanath et al. propose an algorithm that creates an anisotropic quadrilateral mesh by packing rectangular cells [8].

There are several two-dimensional meshing methods that make use of medial axis transformation. Tam and Armstrong [9] propose an algorithm that creates a quadrilateral mesh, and Gürsoy and Patrikalakis [10] propose a method for creating a triangular mesh, both based on medial axis transformation. These methods subdivide the original domain into simple sub-domains, and then mesh each sub-domain. Quadros et al. present another method that uses medial axis transformation [11]. Their algorithm combines medial axis transformation and an advancing front algorithm to create a quadrilateral mesh.

Typically, existing quadrilateral meshing algorithms cannot create a layered quadrilateral mesh of good quality for a narrow two-dimensional domain. They encounter difficulty in the node spacing on the lengthwise boundary curves, Γ_1 and Γ_2. To produce an all-quadrilateral mesh, the number of nodes placed on Γ_1 must be identical to the number of nodes placed on Γ_2. If n nodes are distributed uniformly on Γ_1 and Γ_2, however, distorted elements are commonly generated where the curvature of the boundary changes rapidly. For example, in Figure 2, eleven nodes are uniformly distributed on Γ_1 and Γ_2, but two ill-shaped elements are created where the curvature of the boundary changes rapidly.

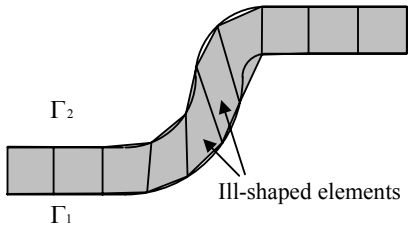


Figure 2 Poorly-shaped elements caused by uniformly distributed boundary nodes

4 Overview of the proposed approach

The proposed method creates a well-shaped layered quadrilateral mesh by using a physically-based one-dimensional meshing method, called bubble packing, and discrete chordal axis transformation [12]. Our method creates a quadrilateral mesh in six steps, illustrated in Figure 3.

Step 1. Discretize the two lengthwise boundary curves, Γ_1 and Γ_2, into a set of line segments.

Step 2. Based on the discretized boundary, create a constrained Delaunay triangulation [13, 14].

Step 3. Extracts a chordal axis, an approximated medial axis, by the algorithm called 'discrete Chordal Axis Transformation' [12].

Step 4. Re-discretize the chordal axis according to the prescribed element length distribution function.

Step 5. Create a single-layered quadrilateral mesh.

Step 6. Convert the mesh into a multi-layered mesh if necessary, depending on the user-specified width threshold.

In steps 1 and 4, we make use of a one-dimensional discretization algorithm called one-dimensional bubble packing. This algorithm packs circular cells, or bubbles, on a given curve, and the bubbles are moved to a stable configuration by dynamic simulation. After the bubbles become stable, their centers give the locations of nodes by which the curve is discretized. The distances between nodes are controlled by a prescribed element length distribution function. By choosing an appropriate element length distribution function for the application, the bubble packing algorithm gives ideal node locations. Section 5 details the bubble packing algorithm.

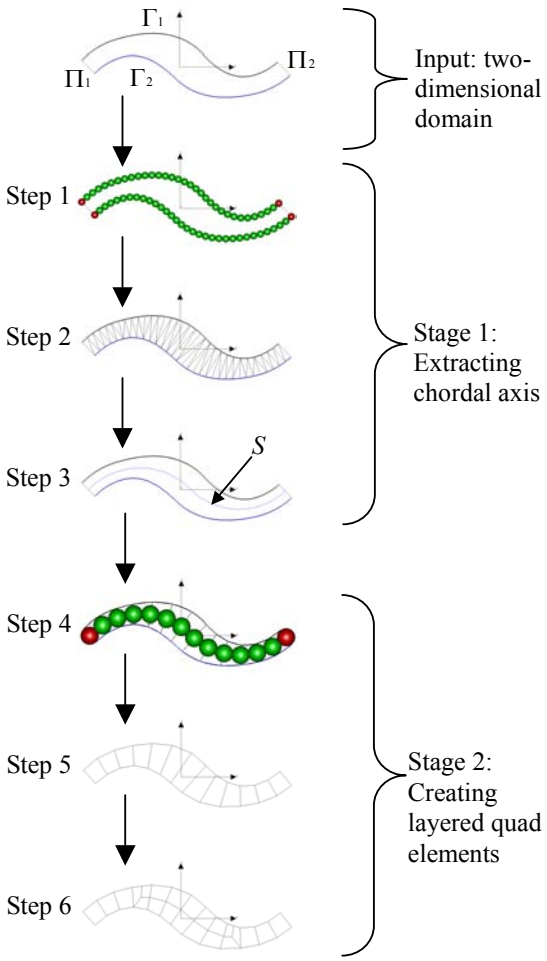


Figure 3 Outline of our approach

Our method has two major stages. The first stage, consisting of Steps 1-3, extracts the chordal axis. The second stage, consisting of Steps 4-6, creates a quadrilateral mesh. We further discuss these stages in Section 6 and Section 7 respectively.

5 One dimensional bubble packing

This section explains a procedure that we call one-dimensional bubble packing. The bubble packing method was originally developed for triangular meshing [15, 16] and was later expanded to anisotropic tetrahedral meshing [17]. For quadrilateral meshing of a narrow two-dimensional domain, we use one-dimensional bubble packing.

The purpose of one-dimensional bubble packing is to find a set of ideally spaced nodes on the boundary curves Γ_1 and Γ_2 in Stage 1 (detailed in Section 6) and on the chordal axis in Stage 2 (detailed in Section 7).

In this process, spherical bubbles are packed on a given curve $C(s)$, $0 \le s \le 1$, and are moved to a stable configuration by dynamic simulation. $C(s)$ can be a series of piecewise line segments, in which case the line segments must be parameterized before the bubble packing process. The diameters of the bubbles are controlled by the element length distribution function $d(C(s))$. If we pack an appropriate number of bubbles closely on the curve with respect to the element length distribution function $d(C(s))$, the bubbles will move to a stable configuration in which all the proximity-based inter-bubble forces are balanced. After the bubbles become stable, their centers give the locations of nodes. Step 1 and Step 4 in Figure 3 show the bubble packing on the boundary curves Γ_1 and Γ_2 and on the chordal axis respectively. In the rest of this section we explain how we simulate the motion of the bubbles.

5.1 Computation of the motion of bubbles

To run the dynamic simulation we have to derive the equation of motion that governs the dynamic behavior of the bubbles. We can then apply a standard numerical integration scheme such as Euler's method or the 4th order Runge-Kutta method to simulate the motion of the bubbles. We derive the second order differential equation that governs the motion of the bubbles by assuming proximity-based inter-bubble forces, a point mass for each bubble, and the effect of viscous damping.

In our method, forces acting on the bubbles are approximated by a mass-spring-damper model. In this model, two kinds of forces act on the bubbles. One is an inter-bubble force analogous to a non-linear spring. The inter-bubble force is a proximity force and acts only between two adjacent bubbles. The other type of force is due to viscous damping.

In the mass-spring-damper model, each bubble has two state variables, position and velocity, and two attributes, mass and a damping coefficient. We denote the state variables of the ith bubble as:

$\mathbf{x}_i$: Position of bubble i

$\dot{\mathbf{x}}_i$: Velocity of bubble i

We assume that all bubbles have the same mass m and the same damping coefficient c, that is:

$$m_i = m$$

$$c_i = c$$

We are now ready to formulate the forces acting on the bubbles.

Figure 4 shows an example of this spring. In order to compute the inter-bubble force between bubble i and bubble j, we need to know the current length and the neutral length of the spring. The current length l of the spring between bubble i and bubble j is computed as:

$$l = \|\mathbf{x}_i - \mathbf{x}_j\|$$

where $\mathbf{x}_i$ is the center of bubble i, $\mathbf{x}_j$ is the center of bubble j, and $\|\cdot\|$ denotes the L_2 Euclidean norm.

We define the neutral length of the spring l_0 as:

$$l_0 = r_i + r_j$$

where r_i is the radius of bubble *i* and r_j the radius of bubble *j*. Because the diameters of the bubbles are controlled by the element length distribution function $d(\mathbf{x})$, l_0 can be written as:

$$l_0 = \frac{d(\mathbf{x}_i) + d(\mathbf{x}_j)}{2}$$

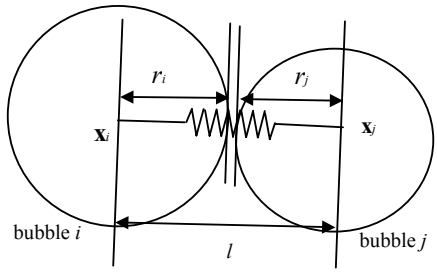


Figure 4 Non-linear spring

After computing l and l_0, the inter-bubble force is computed as a function of the ratio of l and l_0. Two adjacent bubbles either attract each other, repel each other or do not interact depending on the value of l/l_0. When $l/l_0 = 1$, the bubbles are at a stable distance and do not either attract or repel each other. When $l/l_0 < 1$, they repel each other and when $1 < l/l_0 < 1.5$, they attract each other. When $1.5 < l/l_0$, again they do not interact because they are too far away.

As in the previous Bubble Mesh methods, we define the force of the non-linear spring using a cubic function:

$$f(w) = \begin{cases} k(1.25w^3 - 2.375w^2 + 1.125) & \text{for } 0.0 \le w \le 1.5 \\ 0 & \text{otherwise} \end{cases}$$

where w is the ratio of l and l_0, that is $w = l / l_0$, and k is the spring constant.

Since multiple bubbles can be adjacent to a given bubble *i*, the total inter-bubble force acting on the *i*th bubble can be computed by:

$$\mathbf{f}_i = \sum_j f_{ij} \frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|}$$

where

$$f_{ij} = \begin{cases} \text{force applied to bubble } i \text{ by bubble } j & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases}$$

In order to make a system of bubbles converge to a stable configuration, damping must be added to the system. We define a damping force acting on bubble i to be proportional to the velocity of the bubble:

$$-c\dot{\mathbf{x}}_i$$

Finally, we can write down the second order ordinary differential equation that governs the motion of the bubbles as:

$$m\ddot{\mathbf{x}}_i + c\dot{\mathbf{x}}_i = \mathbf{f}_i$$

A numerical integration scheme such as Euler's method or the 4th order Runge-Kutta method can be used to solve this ordinary differential equation.

5.2 Population control

Another important procedure during bubble packing is a process called 'adaptive population control.' Its purpose is to adjust the number of bubbles so that there are no significant gaps or overlaps in the final packed configuration. Bubbles should be closely packed in order to obtain well-distributed nodes. If the number of bubbles is too small, large gaps exist between bubbles, and the distance between bubbles will become larger than the ideal distance specified by $d(C(s))$. In contrast, if the number of bubbles is too large, the bubbles will have large overlaps, making the distance between bubbles shorter than the ideal distance. To obtain well-distributed node points, it is important to adjust the number of bubbles during dynamic simulation.

The optimal number of bubbles is a function of the curve geometry and the specified spacing function, that is:

$$N = N(C, d)$$

where

N	: Appropriate number of bubbles
C	: Target curve
$d = d(C(t))$	: Spacing function

If the curve is straight and if $d(C(s))$ is constant, i.e., $d(C(s)) = d_0$, an appropriate number of bubbles can be estimated by:

$$N = \frac{L(C)}{d_0}$$

where

$L(C)$	: Length of the curve
d_0	: Diameter of a bubble

We can use this formula to estimate an appropriate number of bubbles for a small piece of a curve. Although our program makes use of this formula to make an initial guess of the bubble configuration, when the curve is not straight and $d(C(s))$ is not constant, this formula gives only a rough estimate of an appropriate number of bubbles.

Because our initial guess cannot be accurate, we need to control the number of bubbles adaptively through the packing process. We take advantage of the computation of the inter-bubble force to find over-populated and under-populated regions. If a bubble is receiving repelling forces, this indicates that it is in an over-populated region. In contrast, if

a bubble is receiving attracting forces, this signifies that it is in an under-populated region. Our program deletes bubbles from over-populated regions and adds bubbles in under-populated regions once every a few iterations.

5.3 Constrain bubbles on a curve

Since all nodes must be created on a target curve, the movement of the bubbles must be constrained to the curve. However, because of the curvature a bubble may be pushed away from the curve during dynamic simulation. The inter-bubble force may have a component perpendicular to the tangential direction of the curve. As a result, the bubbles stray away from the curve unless the program explicitly constrains the bubbles on the curve.

In order to constrain all bubbles on a curve, we move bubbles in parametric space instead of in Cartesian space. However, because the force is computed in Cartesian space, the program computes the movement in parametric space approximately from the movement in Cartesian space.

Suppose the i th bubble is located at $\mathbf{x}_i = C(s_0)$ as shown in Figure 5. We define a tangential vector $C^s(s_0)$ as:

$$C^s(s_0) = \begin{pmatrix} \dfrac{\partial C_x(s_0)}{\partial s} \\ \dfrac{\partial C_y(s_0)}{\partial s} \end{pmatrix}$$

We first compute the unconstrained displacement $\Delta\mathbf{x}_i$ in Cartesian space. Then, assuming that the movement is small, such that $C^s(s)$ does not change before and after the iteration, the motion in parametric space Δs is computed approximately as:

$$\Delta s = \frac{v_s}{\left\| C^s(s_0) \right\|}$$

where

$$v_s = \frac{\Delta\mathbf{x}_i \cdot C^s(s_0)}{\left\| C^s(s_0) \right\|}$$

The new position of the bubble, $\mathbf{x}_i'$ is then computed as:

$$\mathbf{x}_i' = C(s_0 + \Delta s)$$

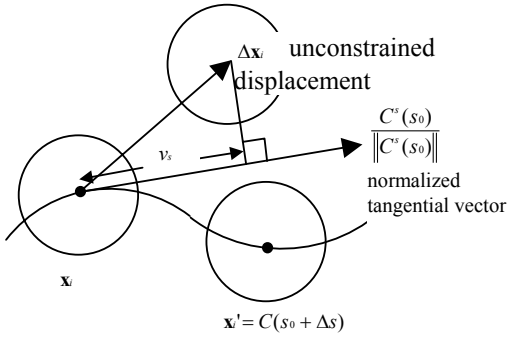


Figure 5 Constrain a bubble on the curve

6 Extraction of a chordal axis

In this section, we explain one of the two major stages of our method, extraction of a chordal axis. A chordal axis, an approximated medial axis, is proposed by Prasad [12]. Our method makes use of his algorithm called 'discrete Chordal Axis Transformation'. Since his algorithm works for a discretized polygonal domain, we first discretize an input domain using one-dimensional bubble packing. Then we construct a constrained Delaunay triangulation, from which we extract the chordal axis.

Discrete Chordal Axis Transformation is simple and efficient. A chordal axis is easily extracted from a constrained Delaunay triangulation, for which many efficient algorithms are readily available.

Our method extracts the chordal axis in the following three steps, illustrated in Figure 3.

(1) Pack bubbles on boundary curves Γ_1 and Γ_2 in order to discretize Γ_1 and Γ_2.

(2) Construct a constrained Delaunay triangulation.

(3) Extract the chordal axis S based on the constrained Delaunay triangulation.

In the first step, bubbles are packed on Γ_1 and Γ_2. Bubbles are moved to a stable configuration by the algorithm described in Section 5. Since the purpose of this step is to find good node locations for a constrained Delaunay triangulation, the diameter of the bubbles must be small enough so that the triangulation approximates the original domain Ω well. In our experiments we chose the diameter of the bubbles to be 1/10 to 1/3 of the minimum width of Ω.

Next, after the bubbles become stable, nodes are placed at the centers of the bubbles and a constrained Delaunay triangulation is constructed. Algorithms to construct a constrained Delaunay triangulation are already available, and they are typically robust and efficient [13, 14].

The chordal axis S is then extracted by joining the centers of edges that connect two nodes, one on Γ_1 and the other on Γ_2. If the bubbles are tightly packed and their

diameters are reasonably small, S will look like the skeleton of the original domain Ω.

We make a slight variation of Prasad's algorithm presented in [12]. In a discretized polygonal domain, Prasad's algorithm connects only the centers of interior edges, i.e., edges that are not a part of a domain boundary. Our chordal axis is slightly different at both ends of the domain. Figure 6 illustrates the difference. Because points on the boundary edges are not included in Prasad's chordal axis, line segments **AB** and **CD** in Figure 6 are not a part of Prasad's chordal axis. In contrast, because our algorithm connects the centers of all the edges that are connecting a node on Γ_1 and a node on Γ_2, line segments **AB** and **CD** are a part of our chordal axis. With this extension, the chordal axis begins at one end of a domain and ends at the other end of a domain. This slight modification is necessary for the next stage of our algorithm, described in the next section.

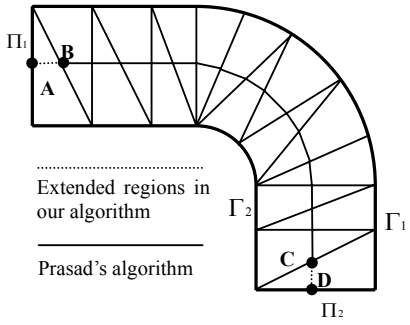


Figure 6 Variation of Prasad's algorithm

7 Creating a layered quadrilateral mesh

After extracting the chordal axis S we create a layered quadrilateral mesh in the following three steps:

(1) Pack bubbles on the chordal axis S complying with a given length distribution function $d(\mathbf{x})$.

(2) Create quadrilateral elements based on the locations of the packed bubbles.

(3) Apply post-processes to refine the elements.

First our program packs bubbles on the chordal axis S as shown in Step 4 in Figure 3. The diameters of the bubbles are controlled according to the size distribution function $d(\mathbf{x})$.

After the bubbles are moved to a stable configuration by the algorithm described in Section 5, quadrilateral elements are created based on the locations of the bubbles. Here, we denote the number of bubbles as n_b and the center of the j th bubble as $\mathbf{x}_j$. Bubbles are sorted by their parametric position on the chordal axis in ascending order. In other words, if a position on the chordal axis is written as $S(s)$, $0 \le s \le 1$, and if the center of the j th bubble in parametric space is $\mathbf{s}_j$, then $s_j < s_{j+1}$.

We define as $L_S(\mathbf{x})$ a straight line that intersects S at point $\mathbf{x}$, and perpendicular to the tangent vector of S at $\mathbf{x}$.

Then nodes $N_{i,j}$, $i = 1,2$, are created based on the location of the j th bubble, as shown in Figure 7. The location of $N_{i,j}$ is the intersection of Γ_i and $L_S(\mathbf{x}_j)$ if $1 < j < n_b$. The location of $N_{i,j}$ is the end point of Γ_i that is closest to the j th bubble if $j = 1$ or $j = n_b$.

After nodes are placed, the j th quadrilateral element is created from $N_{1,j}$, $N_{2,j}$, $N_{2,j+1}$ and $N_{1,j+1}$ as shown in Figure 7.

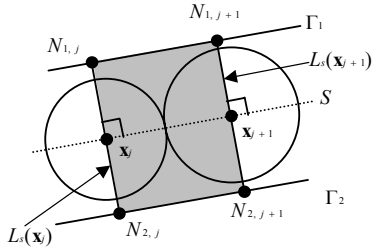


Figure 7 Quadrilateral element generation

After creating elements we apply appropriate post-processes in order to improve the quality of the mesh. If a part of the domain Ω has high curvature, and if the user specified $d(\mathbf{x})$ so that the element length becomes very short, some elements may be inverted as shown in Figure 8. In such a case we apply several iterations of Laplacian smoothing in order to re-distribute the boundary nodes and avoid inverted elements.

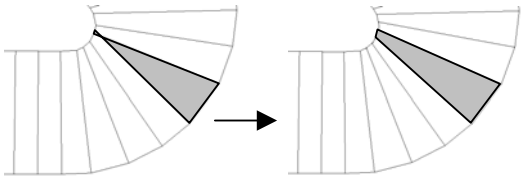


Figure 8 Correcting an inverted element by Laplacian smoothing

If some elements are significantly wider than other elements, they can be converted into two-layers, making the width more uniform. For example, in the mesh shown in Figure 9 (a), elements A, B, C, D and E are wider than other elements. In such a case, we classify each element into three types, (1) narrow element, (2) transition element and (3) wide element as shown in Figure 10. The j th element consisting of nodes $N_{1,j}$, $N_{2,j}$, $N_{2,j+1}$ and $N_{1,j+1}$ is classified by computing l_{ej} and $l_{e(j+1)}$ with l_{thr}, where l_{ej}

is the length of the edge $N_{1,j}\,N_{2,j}$, $l_{e(j+1)}$ the length of the edge $N_{2,j+1}\,N_{1,j+1}$, and l_{thr} a user-specified width-threshold. If both l_{ej} and $l_{e(j+1)}$ are shorter than l_{thr}, the element is classified as a narrow element. If both l_{ej} and $l_{e(j+1)}$ are longer than l_{thr}, the element is classified as a wide element. If only one of l_{ej} or $l_{e(j+1)}$ is longer than l_{thr}, the element is classified as a transition element. In Figure 9 (a) elements B, C and D are classified as wide elements, and elements A and E are classified as transition elements. After classifying elements, we split wide elements into two layers, and we apply the template shown in Figure 9 (b) to transition elements. This yields a partially two-layer mesh as shown in Figure 9 (c).

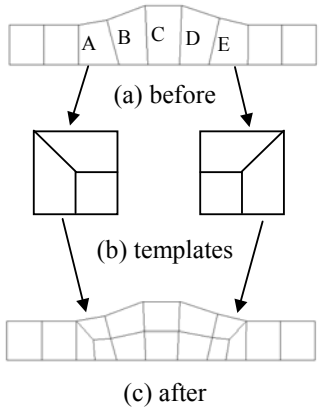


Figure 9 Converting a part of a mesh into two layers

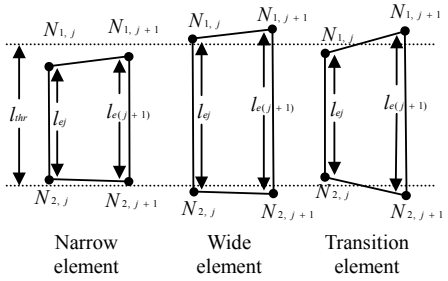


Figure 10 Classification of elements

8 Results

In this section, we present some results of our meshing algorithm and large deformation finite element analysis using our mesh.

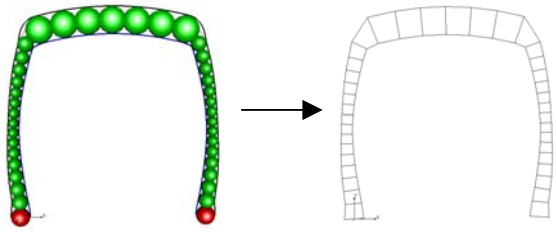
8.1 Tire cross-section

Figure 11 shows three different quadrilateral meshes of a tire cross-section: a single-layer mesh of uniform aspect ratio shown in Figure 11 (a), a single-layer mesh of uniform element length shown in Figure 11 (b), and a partially double-layer mesh of uniform element length and width shown in Figure 11 (c).

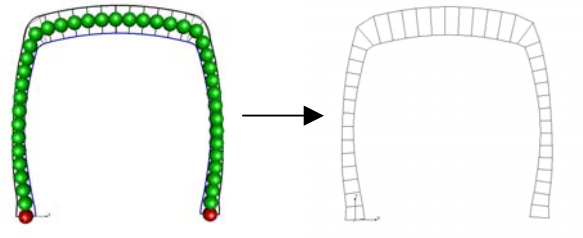
Quadrilateral elements in the mesh shown in Figure 11 (a) have an aspect ratio of almost 1:1. Note that the diameters of bubbles are identical to the width of the domain.

Quadrilateral elements in the mesh shown in Figure 11 (b) have almost uniform length. The element length distribution function $d(\mathbf{x})$ is a constant equal to the smallest width of the cross-section.

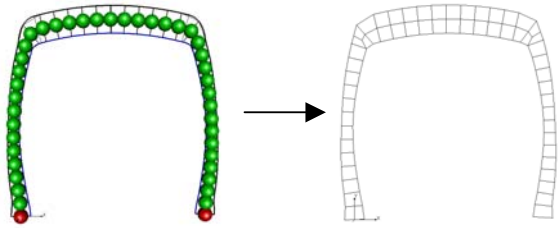
The same length distribution function is used in Figure 11 (c), but the width threshold is chosen so that the mesh becomes partially double-layered.



(a) A single-layer mesh with uniform aspect ratio



(b) A single-layer mesh with uniform element length



(c) A partially double-layer mesh with uniform element length and width

Figure 11 Quadrilateral mesh of a tire cross-section

8.2 Curvature-based length distribution

Figure 12 shows a single-layer mesh with a curvature-based element length distribution function. The element length distribution function is specified so that the length of an element becomes smaller in a region where the curvature of a boundary curve is high. Figure 12 also shows the bubble packing on the chordal axis of the letter m.

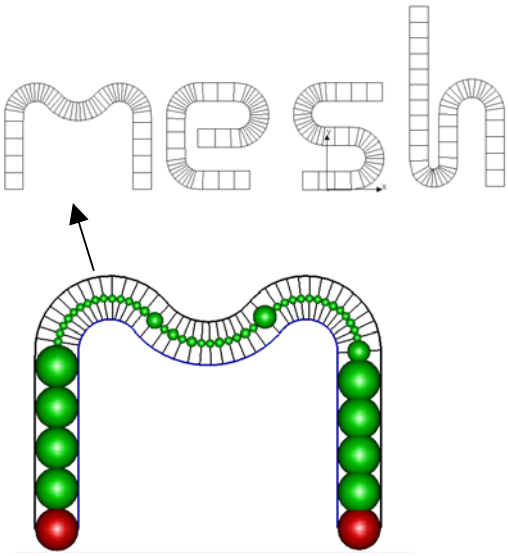


Figure 12 Single-layer mesh with a curvature-based element length distribution function

8.3 Formed metal joint

Figure 13 shows a single-layer mesh of a formed metal joint. This type of joint fastens two sheet metal parts, and the mesh shown is suitable for a finite element analysis that finds how much load this joint can support.

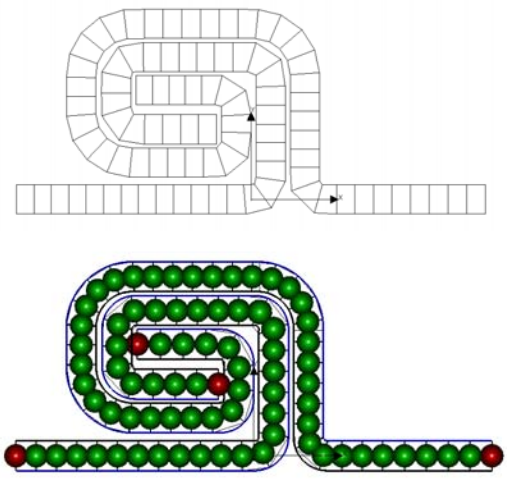


Figure 13 A single-layered mesh of a formed metal joint

8.4 Large deformation FEA

We performed several finite element analyses of a large deformation of a rubber part. The goal of the experiments was to show that the quality of an initial mesh has a significant effect on the accuracy of such a non-linear finite element analysis.

Figure 14 shows experimental results of the finite element analyses. We analyzed the deformation and stress distribution of a rubber part when the partis squeezed by two rigid bodies.

We used three meshes for the experiments:

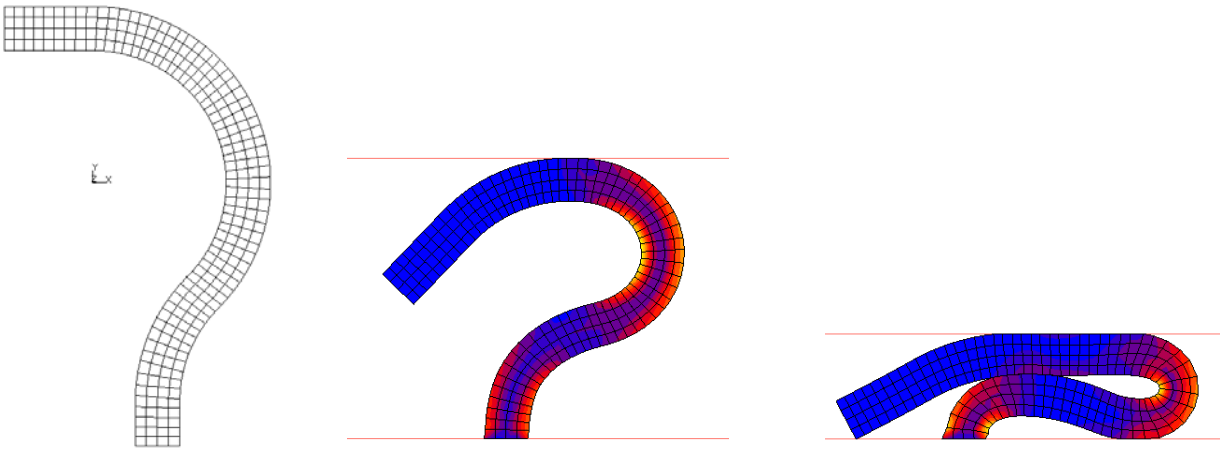
- a four-layer mesh **A** created by our method as shown in Figure 14 (a)
- a single-layer mesh **B** created by our method as shown in Figure 14 (b)
- a single-layer mesh **C** created by a conventional method with uniformly distributed nodes as shown in Figure 14 (c)

The four-layer mesh **A** has 256 elements, and it is reasonable to assume that the solution obtained with this mesh is close to the exact solution.

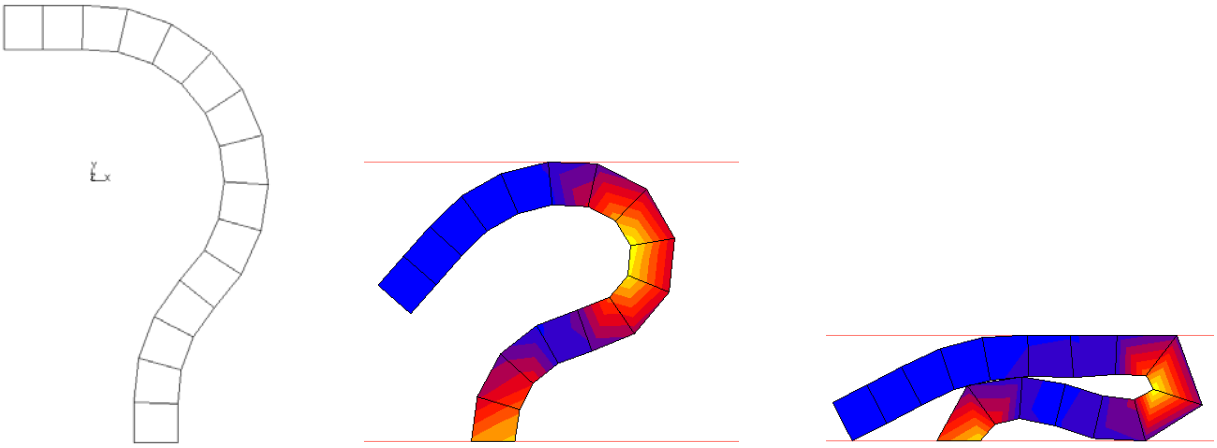
The single-layer mesh **B** has 15 elements, and the mesh shows high quality. Remarkably, none of the elements are inverted anywhere in the domain during or after the deformation.

The single-layer mesh **C** created by conventional method also has 15 elements. But this mesh has some distorted elements. The distortion becomes more severe during the deformation, and finally we see two inverted elements, the 6th and 7th elements from the bottom.

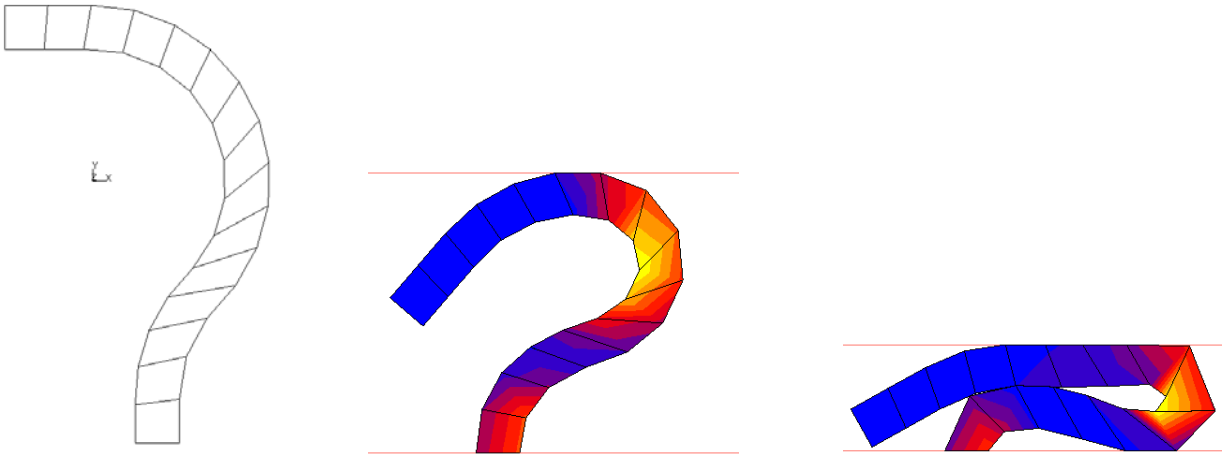
Mesh **B** created by our algorithm shows better results in the maximum Cauchy stress. The maximum Cauchy stress computed with the four-layer mesh is 3.737 N/mm^2. We assume that this value is reasonably close to the exact solution, and we use this value as a reference Cauchy stress. The maximum Cauchy stress computed from single-layer mesh **B** is 3.828 N/mm^2. The error from the reference Cauchy stress is only 2.44%. On the other hand, the maximum Cauchy stress computed with single-layer mesh **C** is 4.31 N/mm^2. The error from the reference Cauchy stress is 15.3%. This result indicates that the quality improvement of a layered quadrilateral mesh by using the proposed approach can improve the accuracy of a large deformation finite element analysis significantly.



(a) Four-layer mesh **A** created by our algorithm. The max Cauchy stress is 3.737 N/mm^2.



(b) Single-layer mesh **B** created by our algorithm . The max Cauchy stress is 3.828 N/mm^2 (2.44% error compared with mesh **A**))



(c) Single-layer mesh **C** created by a conventional meshing algorithm. The max Cauchy stress is 4.310 N/mm^2 (15.3% error compared with mesh **A**)

Figure 14 Large deformation finite element analysis of a rubber part

9 Discussion

The proposed method creates a high quality quadrilateral mesh of a narrow two-dimensional domain. This section discusses the computational cost, the current limitations, and the future extensions of the method.

The major contribution in the reduction of computational time comes from the one-dimensional bubble packing process. The computational time for this process is $O(n)$, where n is the number of bubbles, proportional to the number of elements included in an output quadrilateral mesh. Compared to a general two-dimensional bubble packing scheme, whose computational cost is $O(n \log n)$, therefore, the proposed meshing scheme with chordal axis transformation offers an efficient quadrilateral meshing of a narrow two-dimensional domain.

In the current implementation of the proposed method a case where a chordal axis has branches cannot be handled. If the chordal axis of a given geometry has branches, the domain must be decomposed into a set of sub-domains so that each sub-domain's chordal axis has no branches. While this domain decomposition is not a trivial problem for a general two-dimensional geometry, for a narrow two-dimensional domain it is easier to identify the topological connectivity of branches, and decompose the domain into a set of simpler sub-domains. In a future extension of the proposed method this current limitation will be addressed.

Our experiments indicate that the method creates a high-quality mesh for various geometric domains. There is however no provable bound of the mesh quality—the quality of the mesh depends on the curvature distribution of the domain boundaries as well as the element size distribution function. It is one of our future research topics to identify a set of rules for defining an appropriate element size distribution function for a given curvature distribution.

10 Conclusion

We have presented a new method that automatically creates a layered quadrilateral mesh of a thin, or narrow, two-dimensional domain. The mesh created by our algorithm is appropriate for the finite element analysis of a narrow cross-section with a large deformation.

Our method makes use of a physically-based one-dimensional meshing technique called bubble packing and a technique called discrete Chordal Axis Transformation for extracting a skeleton of a narrow two-dimensional domain.

We have also presented several experimental results. These results indicate that our algorithm creates high quality meshes that are appropriate for finite element analysis of a thin, or narrow, two-dimensional domain.

ACKNOWLEDGMENTS

This material is based in part on work supported under a NSF CAREER Award (No. 9985288).

REFERENCES

1. Blacker, T.D. and M.B. Stephenson, *Paving : A New Approach to Automated Quadrilateral Mesh Generation.* International Journal for Numerical Methods in Engineering, 1991. **32**: p. 811-847.
2. Cass, R.J. and S.E. Benzley, *Generalized 3-D Paving : An Automated Quadrilateral Surface Mesh Generation Algorithm.* International Journal for Numerical Methods in Engineering, 1996. **39**: p. 1475-1489.
3. Owen, S.J., *et al. Advancing Front Quadrilateral Meshing Using Triangle Transformations*. *Proceedings of 7th International Meshing Roundtable*. 1998.
4. Thompson, D.S. and B.K. Soni. *Generation of Quad- and Hex-dominant, Semistructured Meshes Using an Advancing Layer Scheme*. *Proceedings of 8th International Meshing Roundtable*. 1999: Sandia National Laboratory.
5. Shimada, K., J.-H. Liao, and T. Itoh. *Quadrilateral Meshing with Directionality Control through the Packing of Square Cells*. *Proceedings of 7th International Meshing Roundtable*. 1998.
6. Itoh, T., *et al. Automated Conversion of 2D Triangular Mesh into Quadrilateral Mesh with Directionality Control*. *Proceedings of 7th International Meshing Roundtable*. 1998.
7. White, D.R. *Redesign of the Paving Algorithm : Robustness Enhancements through Element by Element Meshing*. *Proceedings of 6th International Meshing Roundtable*. 1997.
8. Viswanath, N., K. Shimada, and T. Itoh. *Quadrilateral Meshing with Anisotropy and Directionality Control via Close Packing of Rectangular cells*. *Proceedings of 9th International Meshing Roundtable*. 2000.
9. Tam, T.K.H. and C.G. Armstrong, *2D Finite element mesh generation by medial axis subdivision.* Advances in Engineering Software, 1991. **13**(5/6): p. 313-324.
10. Gürsoy, H.N. and N.M. Patrikalakis, *An Automatic Coarse and Fine Surface Mesh Generation Scheme Based on Medial Axis Transform: Part I Algorithm, Part II Implementation.* Engineering with Computers, 1992. **8**: p. 121-137,179-196.
11. Quadros, W.R., *et al. LayTracks: A New Approach To Automated Quadrilateral Mesh Generation using MAT*. *Proceedings of 9th International Meshing Roundtable*. 2000.
12. Prasad, L., *Morphological Analysis of Shapes*, in *http://cnls.lanl.gov/Highlights/1997-07/*. 1997, Los Alamos National Laboratory.
13. Shewchuk, J.R. *Triangle: Engineering a 2D Quality Mesh Generator*. *Proceedings of First Workshop on Applied Computational Geometry*. 1996.

14. George, P.L. *TET MESHING: Construction, Optimization and Adaptation*. *Proceedings of 8th International Meshing Roundtable*. 1999.
15. Shimada, K., A. Yamada, and T. Itoh. *Anisotropic Triangular Meshing of Parametric Surfaces via Close Packing of Ellipsoidal Bubbles*. *Proceedings of 6th International Meshing Roundtable*. 1997.
16. Shimada, K. and D.C. Gossard, *Automatic Triangular Mesh Generation of Trimmed Parametric Surfaces for Finite Element Analysis*. Computer Aided Geometric Design, 1998. **15**(3): p. 199-222.
17. Yamakawa, S. and K. Shimada. *High Quality Anisotropic Tetrahedral Mesh Generation via Ellipsoidal Bubble Packing*. *Proceedings of 9th International Meshing Roundtable*. 2000.