The Ball-Vertex Method: a New Simple Spring Analogy Method for Unstructured Dynamic Meshes

Carlo L. Bottasso

Daniel Guggenheim School of Aerospace Engineering, Georgia Institute of Technology, 270 Ferst Dr., Atlanta, GA 30332-0150, USA

Davide Detomi and Roberto Serra

Dipartimento di Ingegneria Aerospaziale, Politecnico di Milano, Via La Masa 34, Milano, 20156, Italy

Abstract

We study the problem of deforming unstructured grids using pseudo-structural lumped-parameter systems. A basic network of edge springs is complemented with an additional set of linear springs that oppose element collapsing. This is here achieved by confining each mesh vertex to its ball, through linear springs that are attached to the vertex and to its projections on the ball entities. The resulting linear pseudo-structural problem can be solved efficiently with an iterative method. The proposed procedures are compared with a revisited version of the torsional spring method with the help of two and three-dimensional example problems.

Key words:
Unstructured deforming meshes; Dynamic meshes; Moving grids; Spring analogy method; Fluid-structure interaction problems.

1 Introduction and Motivation

In this work we consider the problem of deforming unstructured three-dimensional grids. Deforming meshes are used in many computational applications, including problems with moving boundaries and interfaces, and for r

⋆ To appear in *Computer Methods in Applied Mechanics and Engineering.*

adaption. In all these cases, the motion of a portion of the domain boundary is known, and one wants to deform the rest of the mesh in order to accommodate these imposed displacements.

The most commonly used technique for grid deformation is based on the spring analogy method, whose basic idea is to create a network of springs connecting all vertices in the grid. In the first and simplest of this class of methods [2], each mesh edge is replaced by a spring, whose stiffness is inversely proportional to the edge length. This way, longer edges will be softer, while shorter ones will be stiffer, somewhat preventing the collision of neighboring vertices.

While this classical method performs reasonably well in a number of cases, it does indeed fail as soon as the local grid motion is not small compared to the local mesh size. Unfortunately, in many practical cases the necessary grid displacements are not small, for example when conducting implicit coupled fluid-structure interaction simulations. Furthermore, even if the displacements are small, the edge spring method can not prevent the creation of nearly flat elements, since it can not control the collapse mechanisms of the grid elements. When nearly flat elements are produced, the algebraic problem obtained by discretizing the governing PDE's becomes ill-conditioned, which in turn will usually induce severe limitations on the allowable time step size. Furthermore, in the presence of nearly flat elements, edge-vertex (in two dimensions) or face-vertex (in three dimensions) cross-overs are easily obtained. Clearly, any cross-over will lead to a locally invalid grid, and therefore must be avoided.

Suitable procedures have been proposed in the literature to address the limitations of the edge spring method, by reducing the potential for occurrence of cross-overs or delaying the creation of invalid elements. For example, a procedure that tries to separate rigid body motions from deformations was proposed in Reference [8]. However, for some problems deformational motions will still be large even after depuration from the rigid body components. Therefore, there is a need for methods that can specifically deal with large deformation problems. To address this issue, torsional springs were added to the linear edge springs in References [7,5] in order to avoid the possible collapse mechanisms of triangles and tetrahedra.

With the same goal, in this work we propose a new simple method of controlling collapse mechanisms in unstructured triangulations. This method is based on the idea of complementing the linear edge springs with linear face-vertex springs in three dimensions, or linear edge-vertex springs in two dimensions. These additional springs effectively constrain each vertex within the polyhedral ball that encloses it, contrasting the possible collapse mechanisms of the grid elements. Furthermore, the presence of the additional springs is also beneficial in terms of mesh quality, since these springs tend to keep each vertex close to the centroid of the ball, pushing it away from its boundary.

This paper is organized according to the following plan. Definitions, notation and mesh support data structures are introduced in Section 2. Next, the problem of grid deformation is formulated in Section 3, followed by a description of the basic edge spring method in Section 4. Collapse control mechanisms are discussed in Section 5. In §5.1 we describe the proposed method, and we detail the construction of the additional network of springs that complement the basic edge springs. Next, in §5.2 we offer a slightly revisited formulation of the torsional spring method, with the purpose of using it for comparison. The new proposed method is then compared with the edge spring and torsional spring methods on a number of significative two and three-dimensional examples in Section 6. Finally, in Appendices A and B we give further details on the implementation of the proposed method as used for the examples of this paper.

2 Definitions, Notation and Supporting Data Structures

We consider a bounded domain $\Omega \subset \mathbb{R}^d$, with $d \geq 1$, and boundary Γ. $\mathcal{T}_h$ is a simplicial triangulation of $\bar{\Omega}$. A generic simplex K of the triangulation covers a domain $\Omega_K \subset \Omega$ with boundary Γ_K.

The domain Ω can be described by a boundary representation (BRep) of its topology. Topological entities and their mutual relationships can be represented using appropriate data structures. An attribute information associated with each topological entity specifies its geometry: a surface is associated with each face, a curve is associated with each edge, and a point with each vertex. The procedures described in this work are interfaced with a computer aided design (CAD) system, that provides geometric and/or topological interrogations on the nature of the domain Ω through a limited set of functional operators. These operators hide from the present application the details of the particular data structures and implementation used by the CAD system, easing the porting to other systems.

The link to a CAD tool is in general useful to ensure the geometric and topological consistency to the domain Ω of modifications to the local topology and/or geometry of the triangulation $\mathcal{T}_h$. In this work, the topology of the incoming grid is left unaltered, while the procedures modify the mesh geometry by repositioning its vertices. While in a grid deformation problem boundary vertices are in general either fixed or with known imposed displacement, it is sometimes useful, as shown below, to allow the free motion of some of the boundary vertices, that therefore need to "slide" on the domain boundary. Hence, we use the link to a CAD system to ensure the correct placement of these boundary mesh vertices on the true model boundary, even after large motions. This is important for maximum generality of the procedures, and in

particular when dealing with complex, curved models.

Grids too can be represented using a boundary representation, usually in terms of a subset of the four basic topological entities, in the order regions, faces, edges and vertices. For a review of the possible choices, with their relative merits and limitations, see for example Reference [3].

All topological entities in a mesh are generated by discretizing a parent entity in the geometric model. For example, a mesh face can be obtained either by discretizing a model face, or by discretizing a model region. The term "classification" identifies this unique relationship between each entity in the mesh with an entity in the model. The classification information is here required for automating the correct placement of vertices on the model boundary, as previously mentioned, through appropriate queries to the CAD system.

Throughout this work, we use the capital letters R, F, E, V to indicate regions, faces, edges and vertices, respectively. A generic topological entity belonging to one of these four types is labelled T. A simplex, a triangle in two dimensions or a tetrahedron in three, is indicated with the symbol K. Lists of entities are indicated as $\{\cdot\}$; for example, $\{E\}$ denotes a list of edges. $\{T\}_S$ is used to indicate a list of entities in the set S. For example, $\{E\}_{\mathcal{T}_h}$ is the list of all edges in the triangulation $\mathcal{T}_h$.

3 Deforming Meshes

We consider a deforming mesh problem, i.e. a problem where we are interested in deforming the domain Ω, and hence the grid $\mathcal{T}_h$ associated with it, in order to accommodate some given displacement on a portion of its boundary. The basic idea behind all methods for this class of problems is to define suitable fictitious structural properties for the domain. The deformed configuration of the elastic domain can then be computed, under the action of the driving displacements.

The fictitious elastic problem used to compute a deformed grid configuration can be formulated in different ways. First of all, the problem can be regarded as transient or steady. Since the grid deformation problem is completely artificial and no physical solution field is associated with it, it is usually convenient to consider the steady version of the formulation, even when the simulated physical phenomenon is transient. This is also the view adopted in the present work.

Furthermore, the fictitious structural model can be either continuous [9] or discrete [2]. In the former case, the classical partial differential equations of

elastostatics or elastodynamics are discretized in the spatial dimensions, for example using the finite element method. Alternatively, a lumped-parameter discrete structural model can be used. In this work, we consider the discrete case, where the fictitious problem is obtained by defining a suitable network of springs associated with the grid $\mathcal{T}_h$. The problem becomes then how to construct the best possible network of springs that: a) is simple and inexpensive to compute; b) does not contain collapse mechanisms; and c) leads to graded and well shaped deformed grids, even for large imposed displacements.

In general, the domain boundary Γ can be partitioned according to the following criterion:

$$\Gamma = \Gamma_m + \Gamma_0 + \Gamma_s. \tag{1}$$

The moving boundary of the domain Ω is noted Γ_m, and its corresponding discrete version in $\mathcal{T}_h$ is $\Gamma_{m,h}$. On the moving boundary, displacements are known. Several different situations are possible. For example, one might have that

$$\boldsymbol{u} = \boldsymbol{g} \quad \text{on} \quad \Gamma_m, \tag{2}$$

where $\boldsymbol{g}$ represents the prescribed displacement field. More often than not, one will have that only the displacements of the grid vertices in $\Gamma_{m,h}$ are known, i.e.

$$\boldsymbol{u}_i = \boldsymbol{g}_i \quad \forall\, i \in \{V\}_{\Gamma_{m,h}}. \tag{3}$$

Note that in both cases the total displacement is imposed, so that the grid stays attached to the moving boundary. This is however not necessary, and in certain applications one might want to leave more freedom to the grid. For example, one might prescribe only the motion component in the local normal direction to the boundary, therefore letting the mesh vertices free to slide along Γ_m. In this case, one would have

$$\boldsymbol{u} \cdot \boldsymbol{n} = g \quad \text{on} \quad \Gamma_m, \tag{4}$$

where $\boldsymbol{n}$ indicates the local normal to a generic point on Γ. For the sake of brevity, in the following we will consider only the case of Eq (3).

The portion of the boundary where no driving displacements are imposed can be further subdivided into two parts, corresponding to the terms Γ_0 and Γ_s in Eq (1). On Γ_0 the grid displacements are requested to be null:

$$\boldsymbol{u} = 0 \quad \text{on} \quad \Gamma_0. \tag{5}$$

For example, this situation might apply to the case of the far-field boundary in an aeroelastic problem. Finally, on Γ_s displacements are constrained to be tangential to the boundary, i.e.

$$\boldsymbol{u} \cdot \boldsymbol{n} = 0 \quad \text{on} \quad \Gamma_s. \tag{6}$$

In practice, this means that the grid is allowed to slide along the portion Γ_s of the domain boundary. This is typically necessary for an artificial internal boundary as for example a symmetry plane, as exemplified in figure 1, or for free surface problems.

This format of the problem caters for both the steady and the unsteady cases. In the simulation of a transient process, the driving boundary conditions (3) for the mesh deformation problem are to be regarded as functions of time. Consequently, a mesh deformation problem is solved at each time step during the transient simulation.

4 Basic Edge Spring Method

In this section we briefly review the classical edge spring method. Given two vertices, i and j, the edge-vector from i to j is defined as

$$\boldsymbol{e}_{ij} := \boldsymbol{x}_j - \boldsymbol{x}_i, \tag{7}$$

and its length is noted

$$L_{ij} := \sqrt{\boldsymbol{e}_{ij} \cdot \boldsymbol{e}_{ij}}. \tag{8}$$

The unit edge-vector can then be written as

$$\boldsymbol{i}_{ij} = \frac{\boldsymbol{e}_{ij}}{L_{ij}}. \tag{9}$$

The displacements of vertices i and j are noted $\boldsymbol{u}_i$ and $\boldsymbol{u}_j$, respectively. The stretching of the edge spring is now computed as $(\boldsymbol{u}_j - \boldsymbol{u}_i) \cdot \boldsymbol{i}_{ij}$. The resulting force on vertex i is aligned along the unit vector $\boldsymbol{i}_{ij}$ and can be written as

$$\boldsymbol{f}_{ij}^{\text{Edge}} = k_{ij}(\boldsymbol{u}_j - \boldsymbol{u}_i) \cdot \boldsymbol{i}_{ij}\boldsymbol{i}_{ij} = -\boldsymbol{f}_{ji}^{\text{Edge}}, \tag{10}$$

where k_{ij} is the spring stiffness. The spring stiffness is typically chosen as inversely proportional to the edge length,

$$k_{ij} := 1/L_{ij}, \tag{11}$$

so that short edges are stiffer than longer ones, which provides a beneficial effect in the control of the local element deformation.

The position of each vertex is found by writing its equilibrium under the effect of all its n_E edge-connected springs

$$\sum_{j=1}^{n_E} \boldsymbol{f}_{ij}^{\text{Edge}} = 0. \tag{12}$$

The resulting linear system of equations can be solved in a variety of ways, either direct or indirect, and that might require or not the assembly of the stiffness matrix. If assembly is required, the four 3×3 block entry contributions to the stiffness matrix due to the edge connecting vertices i and j are readily found by inspection of Eq (10): $\boldsymbol{K}_{ii} = -k_{ij}\boldsymbol{i}_{ij}\boldsymbol{i}_{ij}^T$, $\boldsymbol{K}_{ij} = \boldsymbol{K}_{ji} = k_{ij}\boldsymbol{i}_{ij}\boldsymbol{i}_{ij}^T$, $\boldsymbol{K}_{jj} = k_{ij}\boldsymbol{i}_{ij}\boldsymbol{i}_{ij}^T$.

In this work we use a Gauss-Seidel iteration to solve the equilibrium problem, as detailed in Appendix B. This approach is commonly used in the literature for solving dynamic grid problems, is simple to implement and does not require the storage of the assembled system of linear equations. However, other methods could clearly be used for the same purpose, a preconditioned conjugate gradient method being a good candidate that will typically outperform the Gauss-Seidel method.

5 Controlling Collapse Mechanisms

The edge spring method contains collapse mechanisms for triangles and tetrahedra. For example, in the three-dimensional case, one can deform an existing tetrahedron to the point of rendering it flat, with only finite stretching of its edges, as shown in figure 2. In order to correct this behavior, suitable additional stiffening devices have to be added to the plain edge spring network, with the final goal of controlling the volume (in three dimensions) or the area (in two) of a grid element. These additional devices should in fact provide an increased stiffness to the pseudo-structural system, that opposes the creation of flat elements. Two such mechanisms are described in the following: the first is a new proposed method that inserts linear springs, while the second is a revisited version of the algorithms of References [7,5] that insert torsional springs.

5.1 Ball-Vertex Springs

In three spatial dimensions, the basic idea of the new proposed method is to introduce additional linear springs that resist the motion of a mesh vertex towards each one of its region-opposed faces. For each region R using a vertex i, a new linear spring is constructed that connects i with its projection p on the plane of the face F_i of R opposite i. This additional spring is exemplified in figure 3 for a single tetrahedron connected to a vertex, for clarity.

Once the additional springs are constructed for all tetrahedra using a vertex i, one has effectively constrained the same vertex not to leave the polyhedral ball

$B_i = \text{ball}(i)$ that encloses it. Given a vertex i, the operator ball(i) first finds all mesh regions that share that vertex, then finds all mesh faces that bound these mesh regions and, among these, discards all of the faces that use vertex i, finally returning all remaining faces. Similarly in two spatial dimensions. B_i is therefore the set of faces (edges, in two dimensions) that encircle the vertex and are one edge away from it.

The resulting mesh repositioning scheme can be straightforwardly modified to handle the two-dimensional problem. In this case, additional linear springs are constructed that connect each mesh vertex with its neighboring face-opposed edges. For the sake of simplicity, we restrict our attention to the sole more interesting three-dimensional problem in the following.

The position of each mesh vertex is found by writing its equilibrium under the combined effect of its edge-connected springs, together with the additional ball-vertex springs. If $\boldsymbol{u}_i$ and $\boldsymbol{u}_p$ indicate the displacements of vertex i and of its projection p on F_i, respectively, the resulting force on i can be expressed, similarly to the edge spring case, as

$$\boldsymbol{f}_{ip}^{\text{Face-Vertex}} = k_{ip}(\boldsymbol{u}_p - \boldsymbol{u}_i) \cdot \boldsymbol{i}_{ip}\boldsymbol{i}_{ip} = -\boldsymbol{f}_{pi}^{\text{Face-Vertex}}, \tag{13}$$

where, exactly as before, $\boldsymbol{e}_{ip} := \boldsymbol{x}_p - \boldsymbol{x}_i$, $L_{ip} := \sqrt{\boldsymbol{e}_{ip} \cdot \boldsymbol{e}_{ip}}$, and $\boldsymbol{i}_{ip} = \boldsymbol{e}_{ip}/L_{ip}$ is the corresponding unit vector. The spring stiffness is chosen here again according to the same criterion, namely $k_{ip} := 1/L_{ip}$.

The only substantial difference with respect to the edge spring case, is given by the fact that $\boldsymbol{u}_p$ in (13) is now the displacement of a virtual point, and not of an existing vertex. One simple way of addressing this problem is by interpolating the displacement at p based on the displacement values at the three face vertices, j, k and l, respectively. First, $\boldsymbol{x}_p$ is computed as the normal projection of i on F_i as

$$\boldsymbol{x}_p = \boldsymbol{x}_i - (\boldsymbol{x}_i - \boldsymbol{x}_j) \cdot \boldsymbol{n}\boldsymbol{n}, \tag{14}$$

where

$$\boldsymbol{n} := \frac{\boldsymbol{i}_{jk} \times \boldsymbol{i}_{jl}}{||\boldsymbol{i}_{jk} \times \boldsymbol{i}_{jl}||} \tag{15}$$

is the unit normal to face F_i. Given $\boldsymbol{x}_p$, the interpolation coefficients ξ and η corresponding to p can be computed such that

$$\boldsymbol{x}_p = \xi\boldsymbol{x}_j + \eta\boldsymbol{x}_k + (1 - \xi - \eta)\boldsymbol{x}_l. \tag{16}$$

No special action is required if the projected point falls outside the targeted face. Finally, given ξ and η, the interpolated displacement at p is obtained as

$$\boldsymbol{u}_p = \xi\boldsymbol{u}_j + \eta\boldsymbol{u}_k + (1 - \xi - \eta)\boldsymbol{u}_l. \tag{17}$$

Even in the case of the ball-vertex springs, the resulting system of linear equations can be solved in a variety of ways. If assembly is required, the 3×3 block entry contributions to the stiffness matrix due to each ball-vertex spring are readily found by inspection of Eq (13) and (17).

In closing this section, we note that it is also possible to use the sole network of vertex-ball springs. This would avoid the computation of the edge springs, and might then imply a somewhat reduced computational cost, at the price of a possibly reduced control on the element deformation.

5.2 Revisited Torsional Springs

In this section we present the torsional spring method of Reference [5], that will be used for comparisons with the new proposed formulation using ball-vertex springs.

In two spatial dimensions, the torsional spring method complements the edge spring network with additional torsional springs at the vertices of each triangle. The possible face collapse mechanism is thereby prevented by controlling the face area. In three spatial dimensions, additional virtual triangular faces are added to each tetrahedron. Each of these additional faces is equipped with torsional springs at its vertices, that oppose the possible face collapse mechanism. In turn, the inserted virtual faces oppose the motion of each vertex of the tetrahedron towards its opposite face, thereby controlling the collapse mechanism of the tetrahedron. Here again, in the following we concentrate on the sole three-dimensional case.

The basic idea of the formulation is to insert, for each region R using a vertex i, a new face F that is perpendicular to face F_i opposite i in R and that has one edge in common with R. Since each vertex in a tetrahedron is used by three edges, one can insert three additional faces for each vertex, for a total of twelve faces per tetrahedron. The additional faces that are inserted in a tetrahedron R for controlling the collapse of vertex i against its opposite face F_i are shown in figure 4. The remaining nine face configurations are obtained by considering the other three tetrahedron vertices. To reduce the computational cost, often one single face per vertex is used, resulting in the insertion of a total of four faces per tetrahedron.

In reality, the request that the inserted face be perpendicular to F_i can lead to very distorted configurations, even for perfectly well shaped tetrahedra. For example, consider figure 5, where a face F is inserted using vertex i and edge ij. It is evident from the figure that the intersection point p_j of the face plane π_F with edge kl can be very far (in the limit, infinitely far) from either vertex k or l, so that the inserted face is extremely skewed. Since very skewed

elements will introduce very large torsional stiffnesses in the mesh, Reference [5] proposed the following remedy to this situation: if point p_j falls outside of edge kl, then it is identified with one of the two bounding vertices of the edge, either k or l (specifically, l in the case of the figure). An alternative way of avoiding this difficulty, is to request that the additional edge ip_j be perpendicular to edge kl, which would lead in the previous example to the more reasonable inserted face F', as depicted in figure 6. This second choice was used for the present work.

With reference to figure 7, consider now the insertion of the additional face F that passes through edge ij. We want to compute the force on vertex i caused by the displacements of the face vertices i, j and p_j. The (small) rotation of edge iq, where $q = j$ or $q = p_j$, in the plane of face F can be expressed as

$$\boldsymbol{\varphi}_{iq} = \frac{1}{L_{iq}} \boldsymbol{i}_{iq} \times (\boldsymbol{u}_q^F - \boldsymbol{u}_i^F). \tag{18}$$

If $q = p_j$, the displacement of the virtual vertex p_j is interpolated based on the displacements of its neighboring vertices, using

$$\boldsymbol{u}_{p_j} = (1 - \xi)\boldsymbol{u}_k + \xi \boldsymbol{u}_l, \tag{19}$$

where ξ is the interpolation coefficient corresponding to point p_j on edge kl. In Eq (18), the displacements producing the edge rotation are to be understood as the displacements of vertices i and q projected onto the plane of face F, i.e.

$$\boldsymbol{u}_i^F = \boldsymbol{P}\boldsymbol{u}_i, \tag{20}$$
$$\boldsymbol{u}_q^F = \boldsymbol{P}\boldsymbol{u}_q, \tag{21}$$

where $\boldsymbol{P}$ is the projector

$$\boldsymbol{P} := \boldsymbol{I} - \boldsymbol{n}\boldsymbol{n}^T, \tag{22}$$

and $\boldsymbol{n}$ is the unit normal to F,

$$\boldsymbol{n} := \frac{\boldsymbol{i}_{ij} \times \boldsymbol{i}_{ip_j}}{||\boldsymbol{i}_{ij} \times \boldsymbol{i}_{ip_j}||}. \tag{23}$$

The edge rotation $\boldsymbol{\varphi}_{iq}$ induces a stretching in the torsional spring at vertex i, whose stiffness is labelled k_i. Similarly, the edge rotation $\boldsymbol{\varphi}_{qi} = \boldsymbol{\varphi}_{iq}$ induces a stretching in the torsional spring at vertex q, whose stiffness is labelled k_q. The resulting moments on the two vertices can therefore be evaluated as

$$\boldsymbol{m}_{iq} = k_i \boldsymbol{\varphi}_{iq}, \tag{24}$$
$$\boldsymbol{m}_{qi} = k_q \boldsymbol{\varphi}_{qi}. \tag{25}$$

The virtual work corresponding to the displacements $\boldsymbol{u}_i^F$ and $\boldsymbol{u}_q^F$ and the

resulting rotations can then be computed as

$$\boldsymbol{m}_{iq} \cdot \boldsymbol{\varphi}_{iq} + \boldsymbol{m}_{qi} \cdot \boldsymbol{\varphi}_{qi} + \boldsymbol{f}_{iq}^{\text{Torsion}} \cdot \boldsymbol{u}_i^F + \boldsymbol{f}_{qi}^{\text{Torsion}} \cdot \boldsymbol{u}_q^F = 0, \tag{26}$$

where $\boldsymbol{f}_{iq}^{\text{Torsion}}$ (respectively, $\boldsymbol{f}_{qi}^{\text{Torsion}}$) is the force on vertex i (respectively, q) due to the stretching of the torsional springs. By inserting now the definition of the torsional moments $\boldsymbol{m}_{iq}$ and $\boldsymbol{m}_{qi}$ in Eq (26), one gets the expression of the force which reads

$$\boldsymbol{f}_{iq}^{\text{Torsion}} = (k_i + k_q)\frac{1}{L_{iq}^2}\, \boldsymbol{i}_{iq} \times \boldsymbol{i}_{iq} \times (\boldsymbol{u}_q^F - \boldsymbol{u}_i^F) = -\boldsymbol{f}_{qi}^{\text{Torsion}}. \tag{27}$$

Clearly, this force lies in the plane of the face. Here again, the various 3×3 block contributions to the stiffness matrix are readily obtained by inspection of Eq (27), together with Eq (19) and (20,21).

The stiffness of the torsional springs at the generic vertex i was computed in Reference [7] as

$$k_i := \frac{1}{\sin^2 \theta_i} = \frac{L_{ij}^2 L_{ip_j}^2}{4A^2}, \tag{28}$$

where θ_i is the face angle at vertex i and A is the area of the inserted face F. This definition of the spring stiffness opposes the creation of very small ($\theta_i \approx 0$) and very large ($\theta_i \approx \pi$) angles, thereby providing a collapse control effect.

However, this choice is not dimensionally consistent with the definition of the edge springs. In fact, since edge spring stiffnesses are inversely proportional to edge lengths, Eq (11), the forces $\boldsymbol{f}_{ij}^{\text{Edge}}$ are non-dimensional quantities. On the other hand, with the definition of the torsional stiffness (28) the forces $\boldsymbol{f}_{iq}^{\text{Torsion}}$ are inversely proportional to a length. Therefore, adding the effect of the torsional springs to that of the edge springs produces a result that is affected by the units of measure; for example, by expressing the same problem in meters or in millimeters, one would change the relative importance of the edge and torsional springs by three orders of magnitude. This is clearly an undesirable effect, that should be avoided. The same undesirable dimensional effect would also be present in a graded mesh, such as for example a mesh for external fluid dynamics problems, where the elements close to the body are typically several orders of magnitude smaller than those in the far field.

There are different possible solutions to this problem. For example, one could define k_i as

$$k_i := \frac{1}{\sin^2 \theta_i}\, \varrho, \tag{29}$$

where ϱ is a local characteristic length, for example the radius of the circumscribed circle of the inserted face or the face average edge. Other choices for the characteristic length ϱ are certainly possible. Alternatively, one could

employ the sole torsional springs [6], avoiding the use of the linear springs altogether. This choice might however imply a somewhat decreased control on the element deformation, especially if the four face configuration is used.

6 Numerical Examples

In this section, two test cases demonstrate the proposed ball-vertex method for moving grids. The new procedure is compared with the revisited formulation of the torsional spring method described in the previous pages.

Mesh quality measures expressed in terms of geometric criteria are used for assessing the characteristics of the two procedures. This gives an objective (i.e. solver and application-independent) comparison of the grids, while actual numerical simulations would raise questions concerning the particular finite element or finite volume formulation, the time stepping algorithm, the boundary conditions used and other problem dependent issues. The use of objective grid quality measures avoids these problems.

We consider quality measures based on the ratio of the radii of the inscribed and circumscribed spheres (or circles in two dimensions), noted r/R, which gives a measure of the stretching of an element, and quality measures based on dihedral angles. Large dihedral angles can negatively impact the solution of partial differential equations, affecting the discretization error and the conditioning of the discrete problem [1]. Very small dihedral angles indicate the generation of sliver elements, which should also be avoided. The smallest dihedral angle between two neighboring mesh faces in the grid is defined as

$$\gamma_{\min} := \min_{K \in \mathcal{T}_h} \min_{E \in \{E\}_K} \gamma_{E,K},$$

where $\{E\}_K$ is the list of edges bounding simplex K, while the largest dihedral angle is

$$\gamma_{\max} := \max_{K \in \mathcal{T}_h} \max_{E \in \{E\}_K} \gamma_{E,K}.$$

The dihedral angle at edge E formed by faces F_1 and F_2 of simplex K is $\gamma_{E,K} = \pi \pm \arccos(\vec{n}_{F_1} \cdot \vec{n}_{F_2})$, being $\vec{n}_F$ the normal to face F. Similarly, we define the average small and large dihedral angles in the grid as

$$\gamma_{\min,\text{avrg}} := \frac{1}{n_K} \sum_{K \in \mathcal{T}_h} \min_{E \in \{E\}_K} \gamma_{E,K},$$

$$\gamma_{\max,\text{avrg}} := \frac{1}{n_K} \sum_{K \in \mathcal{T}_h} \max_{E \in \{E\}_K} \gamma_{E,K},$$

respectively, where n_K is the number of simplices in the grid.

6.1 Two-Dimensional Pitching and Plunging Airfoil

In this first example we consider a pitching and plunging airfoil of unit chord, with pitching amplitude $\theta_0 = 17$ deg and plunging amplitude $h_0 = .22$ chords, that moves according to

$$\theta_n = \theta_0\Big(\cos(2\pi n/N) - 1\Big),$$
$$h_n = h_0\Big(\frac{9}{10}\,\sin(2\pi n/N) - \frac{1}{10}\,\frac{T}{2\pi}\,(\cos(2\pi n/N) - 1)\Big),$$

where n is the generic time step, and $N = 80$ is the number of time steps per cycle. The simulation is conducted for a total of 9 cycles. Figure 8 shows the upper and lower mesh configurations at the end of the first cycle. Displacement at each time step are applied in an incremental fashion, as described in Appendix A, with a maximum number of increments per time step $r_{\max} = 50$. The maximum number of Gauss-Seidel iterations is $k_{\max} = 300$ and the Gauss-Seidel convergence criterion is $\varepsilon = 10^{-5}$.

Figure 9 shows the number of displacement increments at each time step, r_n (see Appendix A), as a function of the time step number, n. Here and in the following, a solid line is used to indicate the results of the proposed ball-vertex method, while a dashed line is used for the revisited torsional spring method. Although the imposed displacements at each time step are rather large, the ball-vertex method exhibits a remarkably constant behavior throughout the simulation. The torsional spring method fails at the end of the fourth cycle, in the sense that the full required displacement was not achieved within the allowed number of displacement increments and Gauss-Seidel iterations.

Quality measures of the grids are examined in the following figures. Figure 10 shows the minimum and average values of the r/R ratio. Notice that both methods are able to maintain a good average quality of the grid, as shown by the two curves in the top part of the figure. This was apparent also in the visual inspection of animations of the grid motion problem: both methods yield deformed grids that appear of nice quality in the "eye norm" and do not seem to degrade during the simulation. However, figure 10 shows that the ball-vertex method seems to have better control on the worst element in the grid. A rapid deterioration of the worst quality measure is clearly evident for the torsional spring method during the fourth cycle, eventually leading to the failure of the simulation.

These effects are confirmed by the plots of the grid angles. In figure 11 we show the behavior of the average small and large angles, $\gamma_{\text{min,avrg}}$ and $\gamma_{\text{max,avrg}}$. The plot shows that the grid quality measures oscillate about a constant value, with no noticeable progressive damaging of the mesh. Figure 12 shows the behavior of the worst angles in the grid, $\gamma_{\min}$ and $\gamma_{\max}$. Here again, oscillations

of constant amplitude throughout the simulation characterize the ball-vertex method. For the torsional spring method, one can notice somewhat larger oscillation amplitudes until, during the fourth cycle, a flat element is generated, as shown by the plot of the smallest angle in the grid.

Notice that we are using fairly large imposed displacements per time step for this example, with the purpose of highlighting possible differences in the behavior of the two schemes in extreme cases. Indeed, as expected, reducing the time step length and therefore reducing the magnitude of the imposed displacements to smaller values, both methods successfully complete the simulation with quite similar results.

6.2 Three-Dimensional Pitching and Plunging Wing

Next, we consider the three-dimensional case of a pitching and plunging wing. A view of the mesh is given in figure 13. The ratio of the largest and smallest elements in the grid is approximatively 10^5 for this mesh, and without the dimensional correction proposed in the previous section for the torsional spring method the algorithm fails at the very first step. For this example, we have used the radius of the circumscribed circle of each inserted face in the definition of the torsional springs, together with the twelve face configuration. The pitching amplitude is $\theta_0 = 20$ deg, while the plunging amplitude is $h_0 = .7$ chords, and

$$\theta_n = \theta_0\Big(\frac{\pi}{180}\Big(-\sin(2\pi n/N + 0.2\pi) + \sin(2.2\pi)\Big)\Big),$$
$$h_n = h_0 \sin(2\pi n/N).$$

The mesh deformation problem is solved for 4 cycles, using N=800 time steps per cycle. The Gauss-Seidel convergence criterion is $\varepsilon = 10^{-5}$.

Figure 14 shows the number of incremental steps, r_n, as a function of the time step number. Here again, a solid line indicates the proposed ball-vertex method, while a dashed line corresponds to the results of the torsional spring method. On average, fewer incremental steps are necessary for the former approach, which also shows a remarkably constant behavior. This is important, since it means that no deterioration of the grid quality is produced throughout the simulation.

Figure 15 shows the r/R ratio. The average quality of the grid oscillates about a constant value for both methods. A flat element is produced by the torsional spring method during the first cycle. However, contrary to the previous example, the simulation was continued in this case even after the creation of an invalid element, in order to be able to compare the average grid quality. Fig-

ure 16 shows the average small and large dihedral angles, $\gamma_{\text{min,avrg}}$ and $\gamma_{\text{max,avrg}}$. Both quantities oscillate about constant values and do not show progressive damaging of the mesh. Finally, figure 17 reports the worst small and large dihedrals, and shows the rapid deterioration of the grid in the torsional spring case in the first cycle. All the results for the three-dimensional case seem to closely parallel those found for the simpler two-dimensional problem.

7 Conclusions

In this work we have presented a new method for enhancing the robustness of pseudo-structural techniques for the deformation of unstructured meshes. Similarly to the torsional springs method, the pseudo-structural system is designed in such a way as to avoid the appearance of collapse mechanisms for the mesh elements.

Control on the volume (or area) of a grid element is here achieved by confining each vertex to its ball. Additional linear springs are added to the network; these springs connect each vertex with additional virtual points obtained as projections on the ball entities, either edges or faces in, respectively, two and three dimensions. The method is straightforwardly implemented and integrated with the edge spring method; alternatively, it could be used by itself, with a slightly lower computational cost and a slightly decreased control on the element deformation.

The new procedures were compared with the torsional spring method. Numerical experiments in two and three dimensions have shown that the new method behaves well, and is able to guarantee nicely graded, good quality grids even after repeated cycles of fairly severe deformations. For the torsional spring method, a minor but eventually important modification to the original formulation was proposed to make it dimensionally consistent; nevertheless, the revisited torsional springs would seem to be somewhat less robust than the new proposed procedure, at least for the problems here considered. In any case, the proposed ball-vertex method would seem to offer a simple and straightforward solution to the problem of devising robust, effective, pseudo-structural discrete systems for mesh deformation problems, especially in three spatial dimensions.

References

[1] I. Babuska and A.K. Aziz, On the angle condition in the finite element method, *SIAM Journal on Numerical Analysis* **13** (1976) 214–226.

[2] J.T. Batina, Unsteady Euler airfoil solutions using unstructured dynamic meshes, AIAA Paper No. 89-0150, AIAA 27th Aerospace Sciences Meeting, Reno, NV, USA (1989).

[3] M.W. Beall and M.S. Shephard, A general topology-based mesh data structure, *International Journal for Numerical Methods in Engineering* **40** (1997) 1573–1596.

[4] C.L. Bottasso and D. Detomi, A procedure for tetrahedral boundary layer mesh generation, *Engineering with Computers* **18** (2002) 66–79.

[5] C. Degand and C. Farhat, A three-dimensional torsional spring analogy method for unstructured dynamic meshes, *Computers and Structures* **80** (2002) 305–316.

[6] C. Farhat, Personal communication, 2003.

[7] C. Farhat, C. Degand, B. Koobus and M. Lesoinne, Torsional springs for two-dimensional dynamic unstructured fluid meshes, *Computer Methods in Applied Mechanics and Engineering* **163** (1998) 231–245.

[8] C. Farhat, K. Pierson and C. Degand, Multidisciplinary simulation of the maneuvering of an aircraft, *Engineering with Computers* **17** (2001) 16–27.

[9] T.E. Tezduyar, M. Behr and J. Liou, A new strategy for finite element computations involving moving boundaries and interfaces – the deforming-spatial-domain/space-time procedure: I. The concept and the preliminary tests, *Computer Methods in Applied Mechanics and Engineering* **94** (1992) 339–351.

Appendix A Incremental Displacement Algorithm

In this section we describe an incremental procedure for the solution of the mesh deformation problem. A possible implementation of the algorithm is given in figure 18.

The current position vector of vertex i of the incoming grid $\mathcal{T}_h$ is noted $\boldsymbol{x}_i$. The moving boundary Γ_m is discretized in $\mathcal{T}_h$ as $\Gamma_{m,h}$. The prescribed boundary conditions on the discrete moving boundary can therefore be expressed as

$$\boldsymbol{u}_i = \boldsymbol{g}_i \quad \forall\, i \in \{V\}_{\Gamma_{m,h}}, \tag{30}$$

where $\{V\}_{\Gamma_{m,h}}$ is the set of moving vertices.

The algorithm operates as follows. All lumped parameters that are necessary for constructing the edge springs, the additional collapse-avoidance springs and their associated data are computed as functions of the current configuration, $\boldsymbol{x}_i \forall i \in \mathcal{T}_h$, and stored for later use. Next, an iteration is started, with iteration

index r. At the current iteration r, the displacement scaling factor $\alpha^r \leq 1$ is computed. α^r is used to compute the displacement increment for each moving vertex that will be applied at the current iteration. This is given as

$$\boldsymbol{u}_i^r = \alpha^r \boldsymbol{g}_i, \quad \forall\, i \in \{V\}_{\Gamma_{m,h}}. \tag{31}$$

The scaling factor α^r is computed in the following fashion. For each vertex i on the moving boundary $\Gamma_{m,h}$, the list of faces (edges, in two dimensions) B_i is computed as

$$B_i := \text{ball}(i). \tag{32}$$

The minimum distance d_i^r between vertex i and each face (edge, in two dimensions) $F \in B_i$ is then computed as

$$d_i^r := \min_{F \in B_i} \text{dist}(\boldsymbol{x}_i^r, F). \tag{33}$$

The quantity d_i^r can then be interpreted as the radius of the ball inscribed sphere (circle, in two dimensions). Based on d_i^r, a scaling factor for the applied displacement at vertex i at the current iteration can be computed as

$$\alpha_i^r := \left(\frac{d_i^r}{s} + u_i^{r-1} \right) \frac{1}{g_i}, \tag{34}$$

where s is a safety factor, typically chosen as $s = 2$ in this work, $u_i^{r-1} := \sqrt{\boldsymbol{u}_i^{r-1} \cdot \boldsymbol{u}_i^{r-1}}$ is the magnitude of the applied displacement at the previous iteration, and $g_i := \sqrt{\boldsymbol{g}_i \cdot \boldsymbol{g}_i}$ is the magnitude of the desired full applied displacement. In practice, α_i^r represents the safe scaling factor for vertex i at the current iteration which ensures that the corresponding incremental displacement will not locally entangle the grid. Finally, the global scaling factor, α^r, is chosen as the minimum of all scaling coefficients computed for the various moving vertices, or it is set to one once the full required displacement has been reached, i.e. α^r is computed as

$$\alpha^r := \min\left(1, \min_{i \in \{V\}_{\Gamma_{m,h}}} \alpha_i^r\right). \tag{35}$$

Based on the computed value α^r, the imposed displacements at iteration r are evaluated by means of Eq (31). At this point, the linear problem resulting from the spring analogy method is solved, to yield the resulting displacements $\boldsymbol{u}_i^r$ for all vertices i in the grid that are not on $\Gamma_{m,h}$ or $\Gamma_{0,h}$. The increment iteration is completed by updating all position vectors in the grid, except for those on $\Gamma_{0,h}$. Iterations are terminated when $\alpha^r = 1$, i.e. when the full displacements have been applied.

The algorithm is completed by projecting all vertices classified on Γ_s on the corresponding model entities. The projection enforces the sliding constraints (6) on these portions of the boundary. This is obtained in the algorithm of figure 18 through the closest-point operator

$$\boldsymbol{x}' = \mathcal{P}(T, \boldsymbol{x}), \tag{36}$$

which takes as input the model entity T where snapping should take place and an initial guess $\boldsymbol{x}$, and returns a snapped position $\boldsymbol{x}'$ that lies on T.

Non-linear versions of the iterative scheme are possible, for example by updating the spring stiffnesses and accompanying data during the iteration process. This strategy however comes at the price of an increased computational cost, and it is therefore not adopted here.

Appendix B Gauss-Seidel Solution

At the k-th Gauss-Seidel iteration, each vertex i is visited and its equilibrium is enforced. For example, using the sole edge springs, Eq (12), we have

$$\left(\sum_{j=1}^{n_E} k_{ij}\boldsymbol{i}_{ij}\boldsymbol{i}_{ij}^T\right)\boldsymbol{u}_i^{[k]} = \sum_{j=1}^{n_E} k_{ij}\boldsymbol{i}_{ij}\boldsymbol{i}_{ij}^T\boldsymbol{u}_j^{[k]}, \tag{37}$$

where the displacement $\boldsymbol{u}_i^{[k]}$ is unknown and is computed as a function of the known current displacements $\boldsymbol{u}_j^{[k]}$.

As stopping criterion for the Gauss-Seidel iterations, we use the following expression:

$$\max_{i\in\{V\}_{\mathcal{T}_h}} \frac{d_i^{[k]}}{d_i^r} \leq \varepsilon, \tag{38}$$

where ε is a convergence tolerance, $d_i^{[k]} = \sqrt{\boldsymbol{d}_i^{[k]} \cdot \boldsymbol{d}_i^{[k]}}$, and $\boldsymbol{d}_i^{[k]}$ is the local displacement at the current Gauss-Seidel iteration, i.e.

$$\boldsymbol{d}_i^{[k]} := \boldsymbol{u}_i^{[k]} - \boldsymbol{u}_i^{[k-1]}. \tag{39}$$

The quantity $d_i^{[k]}/d_i^r$ represents the ratio of the current displacement to the radius of the ball (see Appendix A, Eq (33)) inscribed sphere or circle. This is a local (vertex based) criterion, in the sense that it measures the actual displacement scaled by the value of the local ball size. This criterion is better suited to graded meshes, that can exhibit changes in the local ball size of several orders of magnitude, than a global stopping criterion, such as for example $\max_{i\in\{V\}_{\mathcal{T}_h}} d_i^{[k]} \leq \varepsilon$.

For efficiently propagating the information during the Gauss-Seidel solution process, vertices need to be ordered. In fact, for reducing the number of iterations, we would like to maximize for each vertex the number of already processed neighbors in order to quickly propagate the information from the moving boundary to the rest of the grid.

A simple ordering scheme for this goal was proposed in Reference [4]. First, model faces classified on Γ_m are collected in connected sets, i.e. if two model faces share a common model edge or vertex they are assigned to the same set. Next, a mesh vertex classified on model boundary is selected for each connected set. This vertex represents the "seed" point for an ordering process. The seed vertex is assigned the number 1 and labelled V_1. Starting from V_1, all edge-connected vertices that are also classified on the connected set or its boundaries are gathered, numbered and stored in a linked list. The vertex that was assigned the number 2, V_2, is now considered. The process is repeated, and all edge-connected vertices to V_2 that are also classified on the connected set or its boundaries are collected in the list and numbered. At the end of this first phase, all vertices in $\Gamma_{m,h}$ have been numbered. The process is now restarted from vertex V_1. All edge-connected vertices that have not yet been numbered are gathered, numbered and stored in the linked list. Next, vertex V_2 is considered, and all its edge-connected vertices that have not yet been numbered are processed. The procedure continues until all vertices are numbered.

The algorithm essentially constructs successive layers of vertices, the first layer being represented by all vertices belonging to $\Gamma_{m,h}$, that wrap the connected set and march away from it. The procedure is illustrated in figure 19.

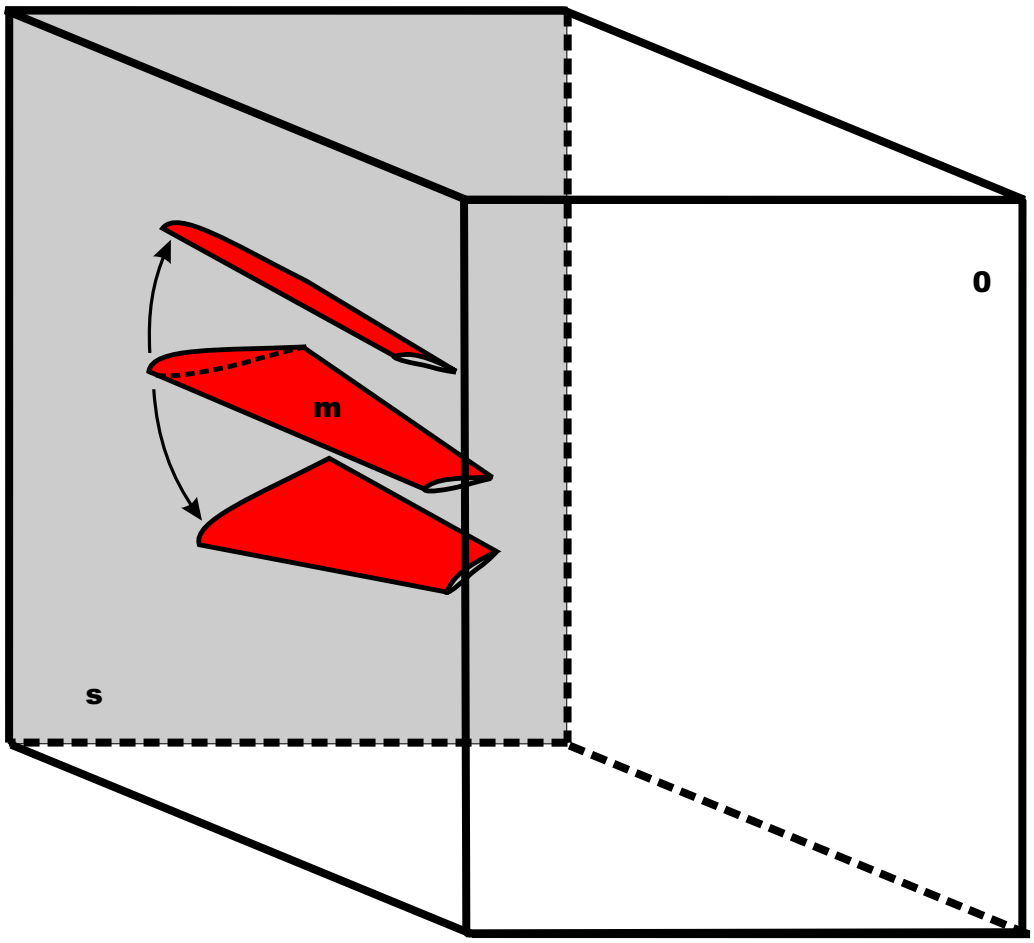

Figure 1. Partitioning of domain boundary. Moving boundary: Γ_m; null displacement boundary: Γ_0; sliding boundary: Γ_s.

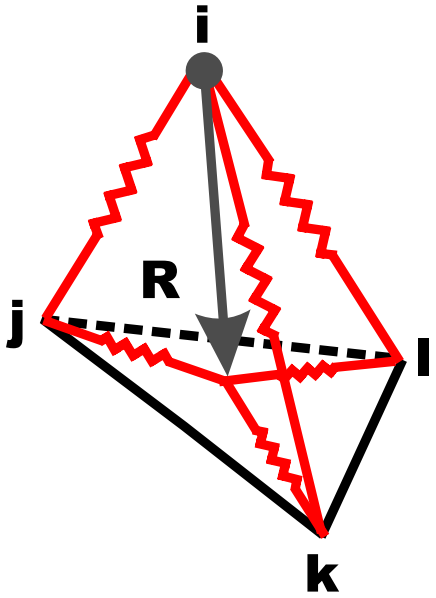

Figure 2. Tetrahedron collapse mechanism with the edge spring method. A vertex can cross one of the faces of its ball with only finite stretching of its connected edge springs.

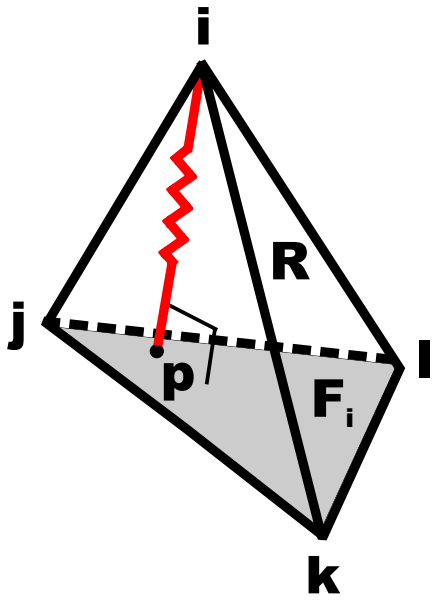

Figure 3. Ball-vertex spring method: additional linear spring connecting each vertex i with the opposite face F_i in a tetrahedron R.

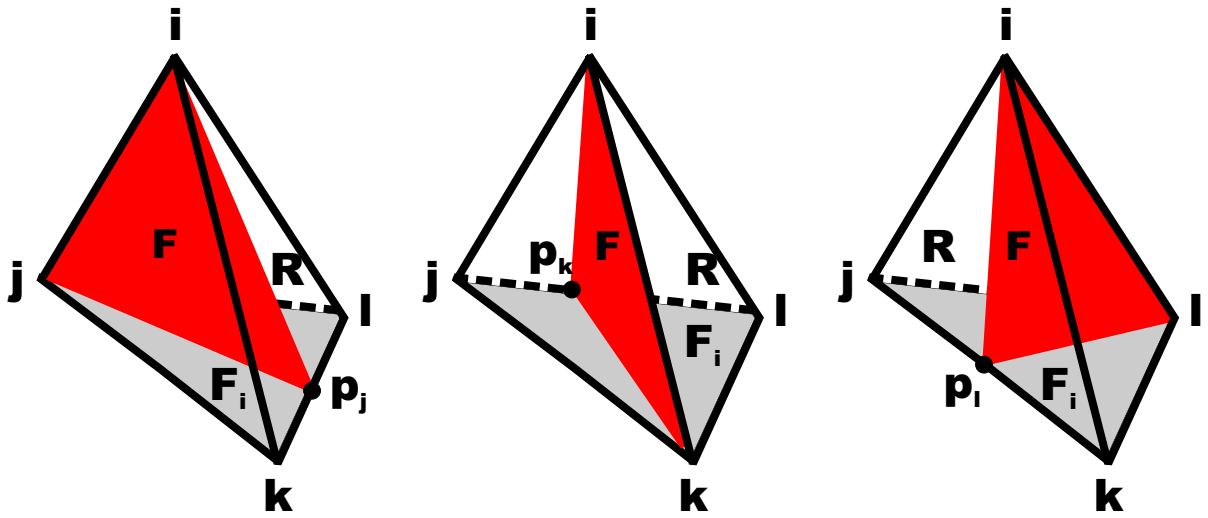

Figure 4. Torsional spring method: additional faces inserted in a tetrahedron R for controlling the collapse of vertex i onto face F_i.

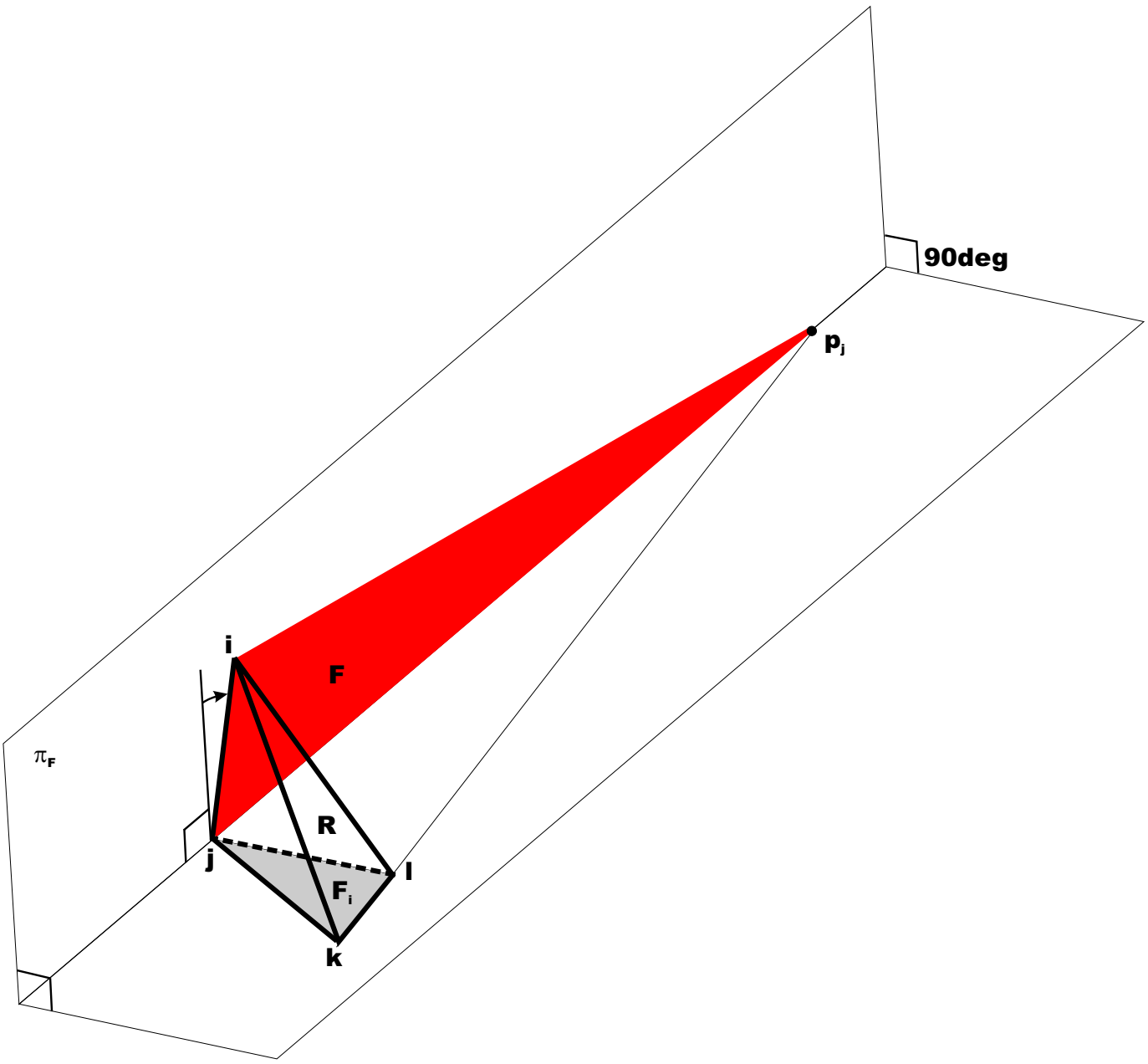

Figure 5. Torsional spring method: distorted additional face for a well shaped tetrahedron.

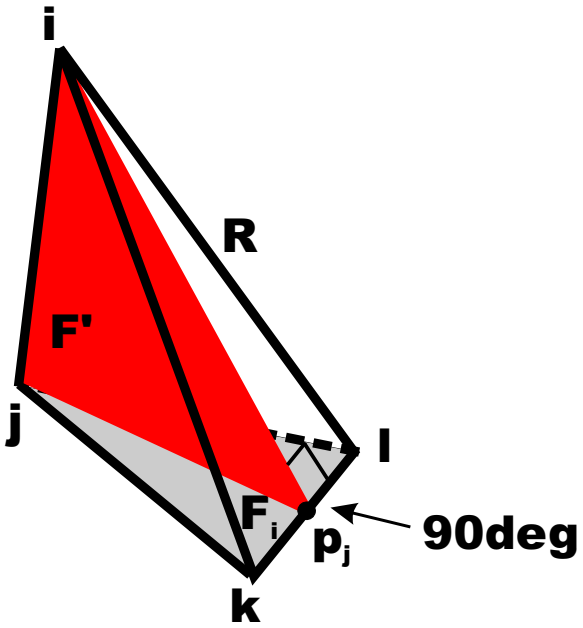

Figure 6. Torsional spring method: modified face insertion criterion.

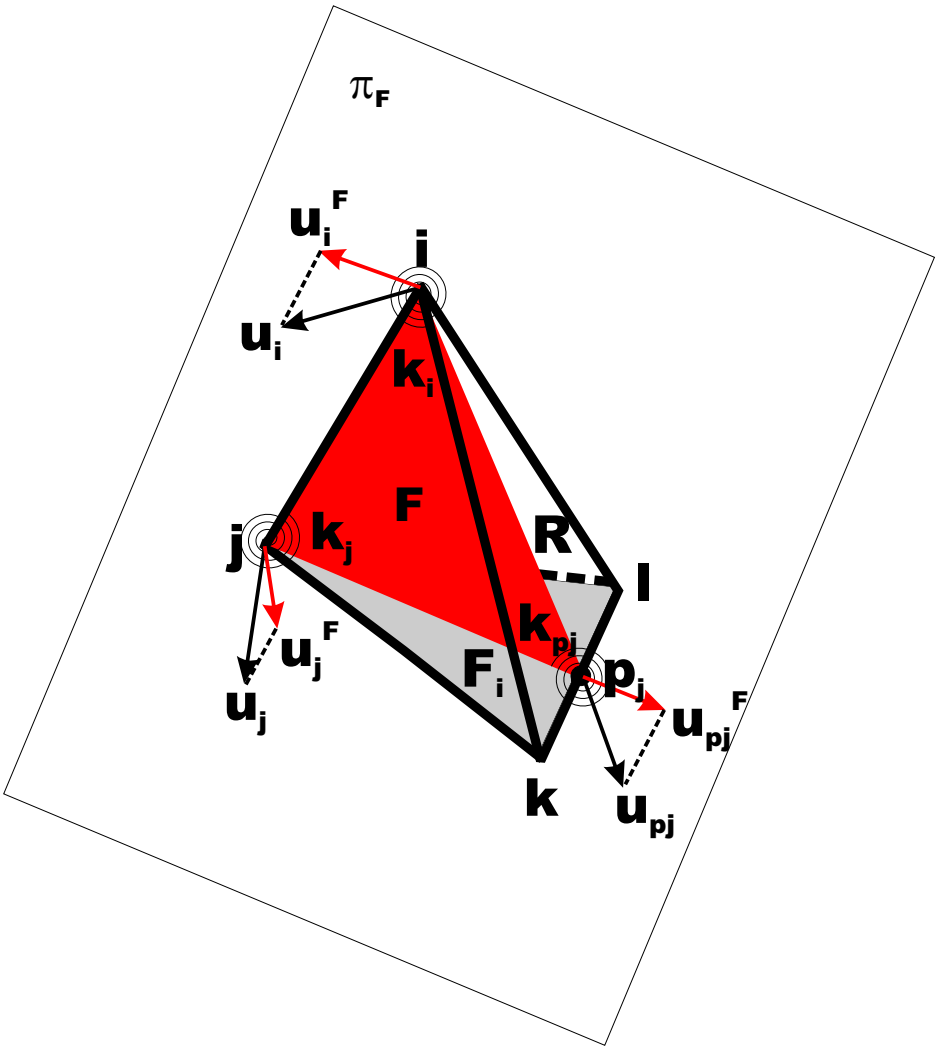

Figure 7. Torsional spring method: inserted face and related quantities.

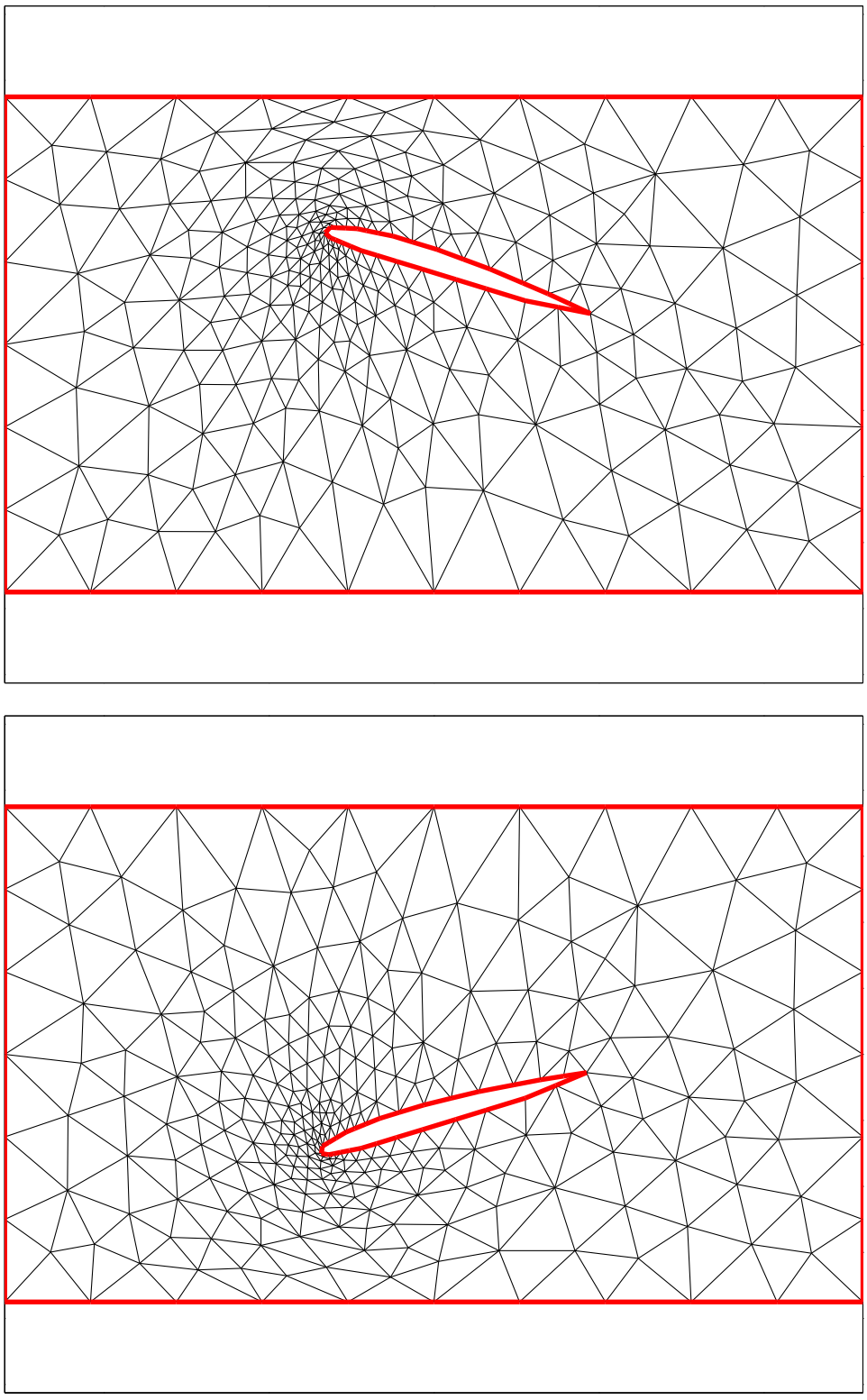

Figure 8. Pitching and plunging airfoil: upper and lower mesh configurations during the first cycle.

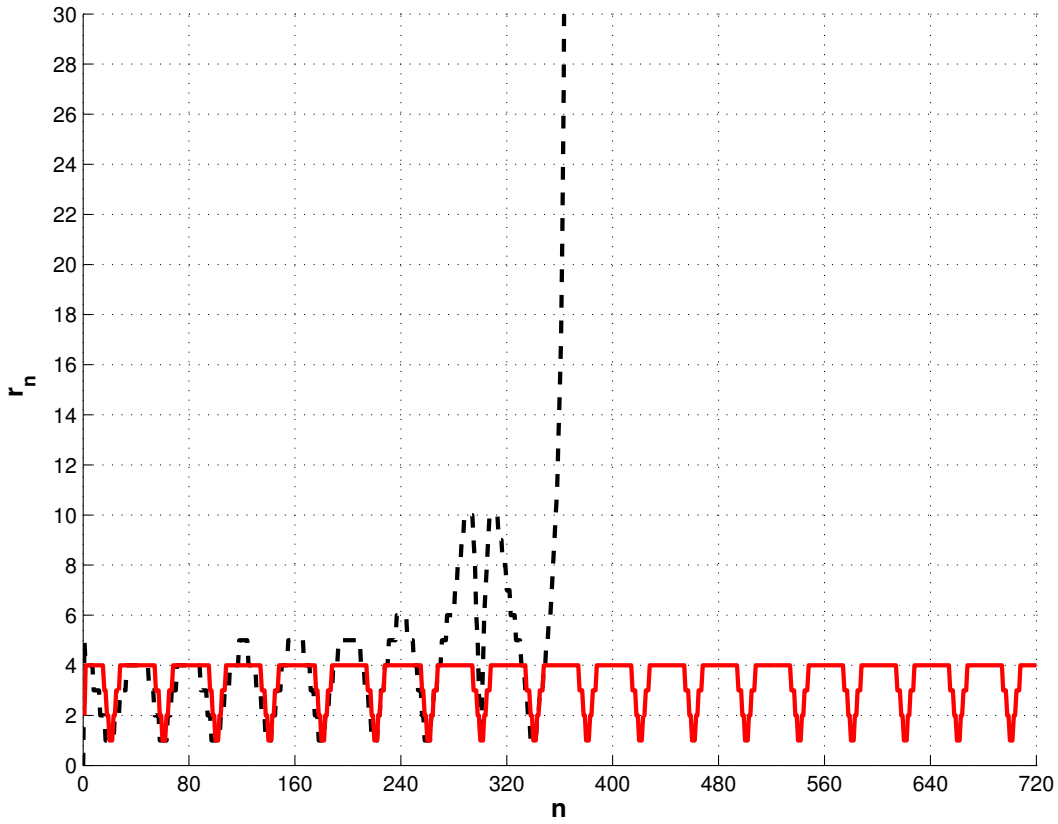

Figure 9. Pitching and plunging airfoil: number of incremental steps r_n as a function of the time step number n. Solid line: ball-vertex method; dashed line: torsional spring method.

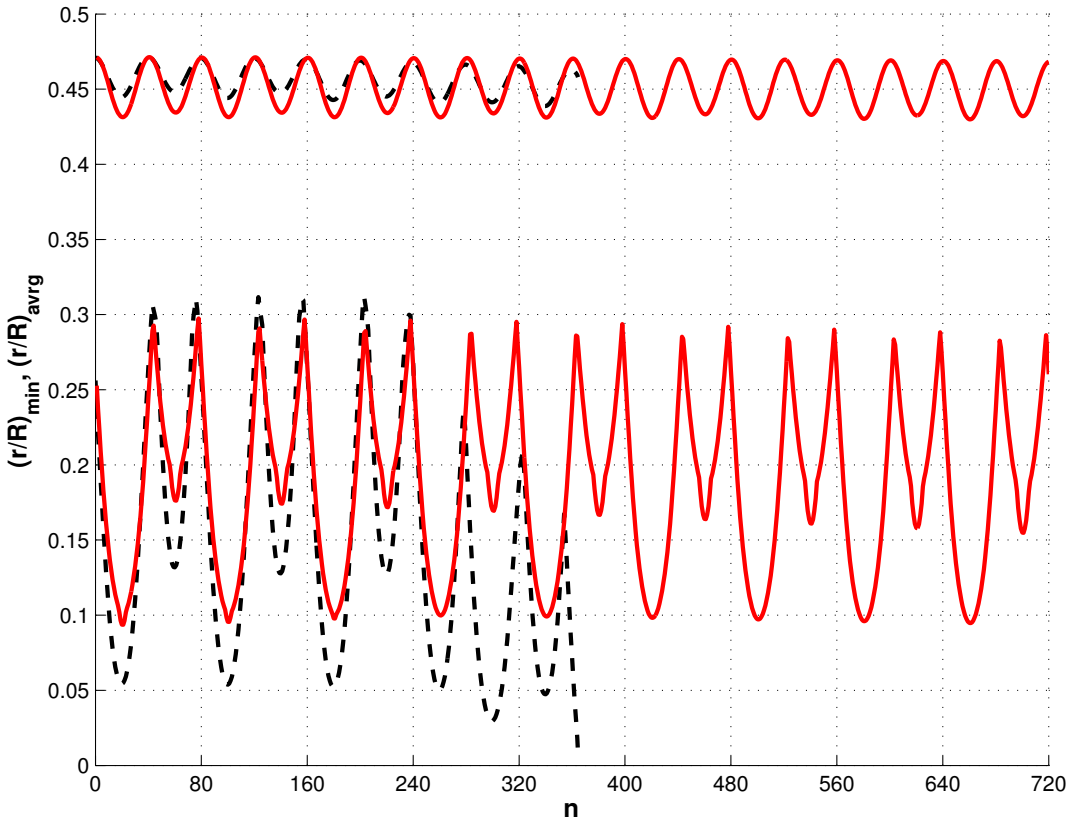

Figure 10. Pitching and plunging airfoil: minimum, $(r/R)_{\min}$, and average, $(r/R)_{\text{avrg}}$, radii ratios as functions of the time step number n. Solid line: ball-vertex method; dashed line: torsional spring method.

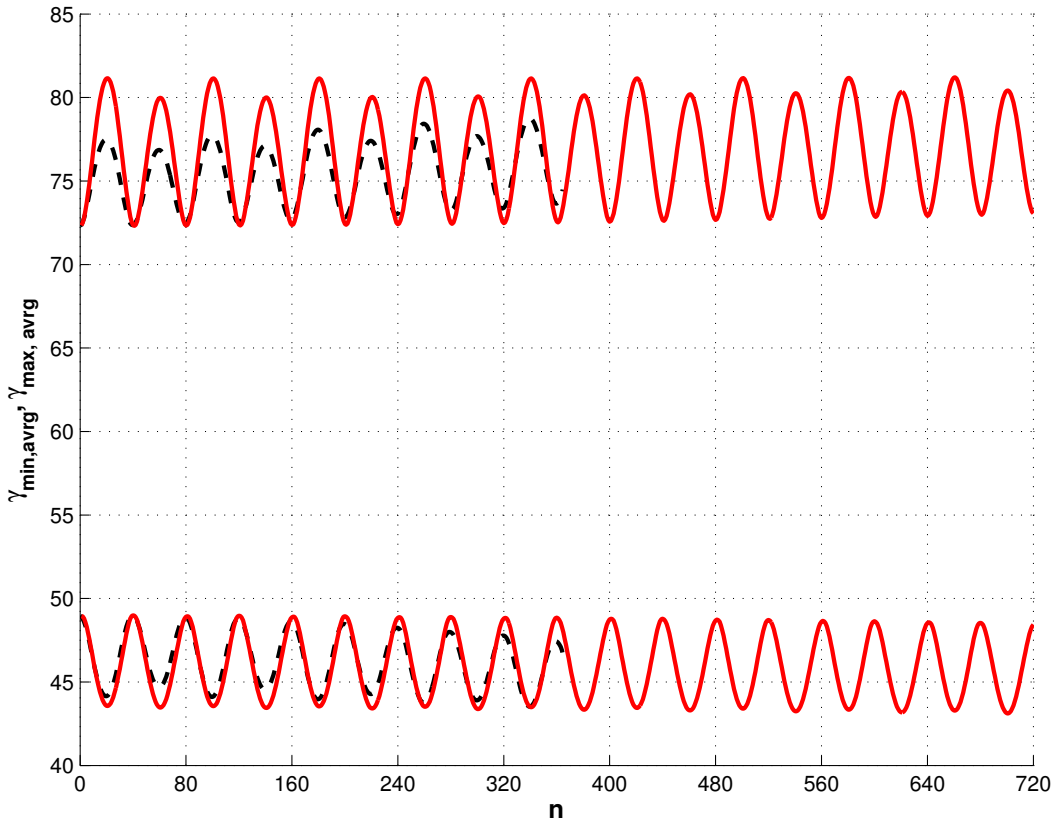

Figure 11. Pitching and plunging airfoil: average minimum, $\gamma_{\mathrm{min,avrg}}$, and average maximum, $\gamma_{\mathrm{max,avrg}}$, angles as functions of the time step number n. Solid line: ball-vertex method; dashed line: torsional spring method.

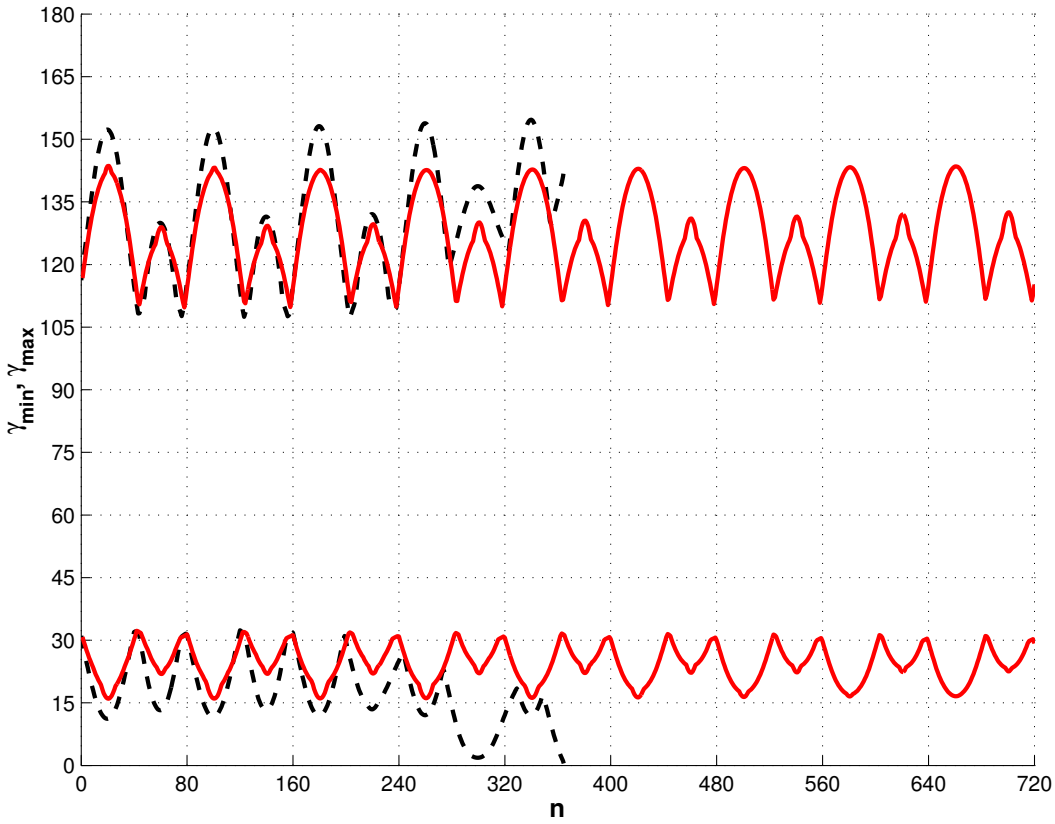

Figure 12. Pitching and plunging airfoil: worst minimum, $\gamma_{\min}$, and worst maximum, $\gamma_{\max}$, angles as functions of the time step number n. Solid line: ball-vertex method; dashed line: torsional spring method.

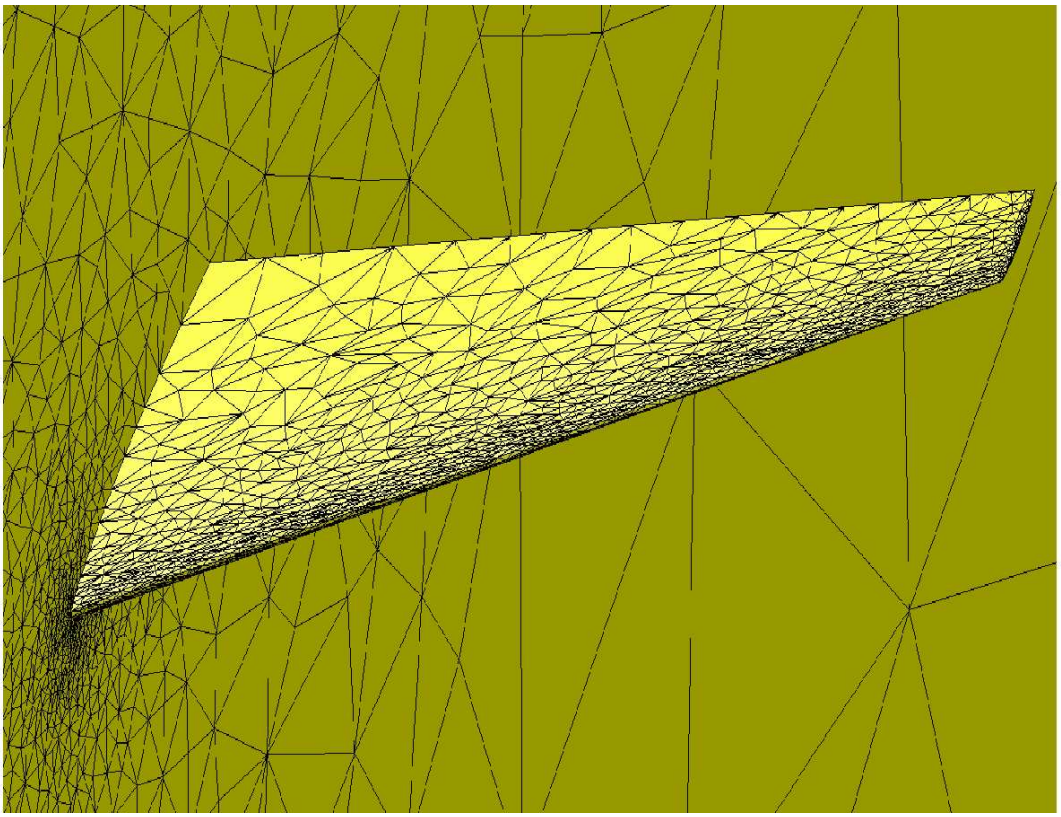

Figure 13. Pitching and plunging wing: view of the initial grid.

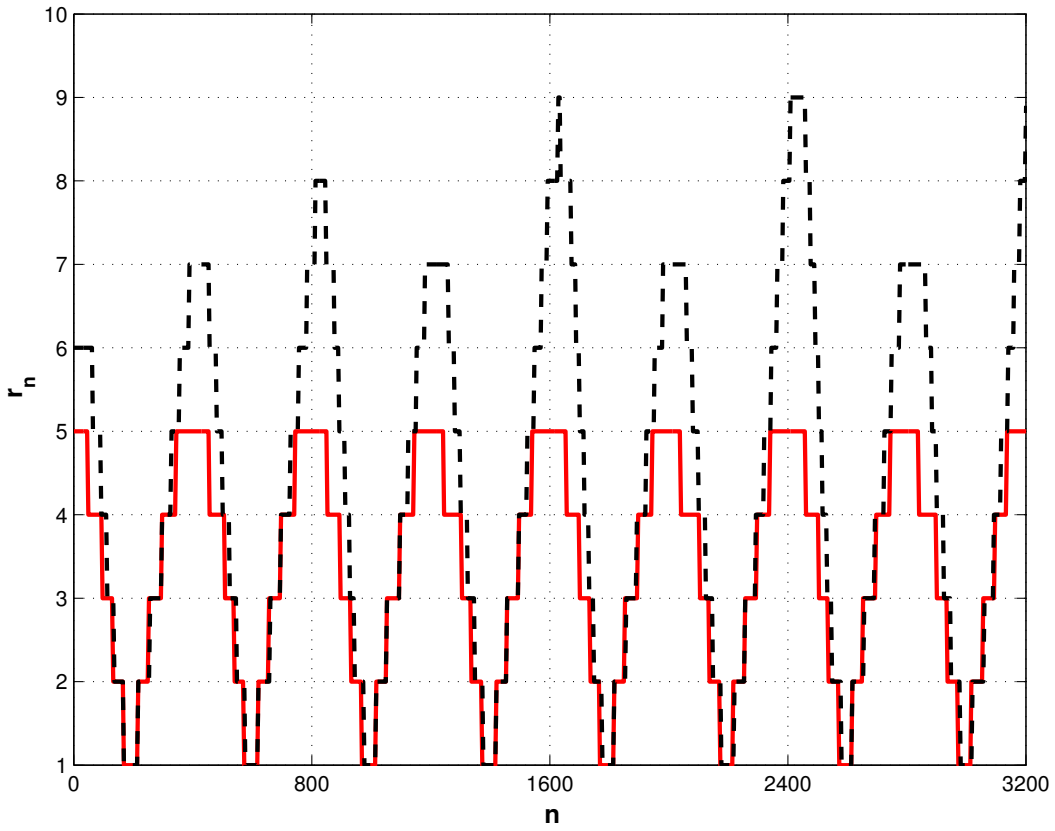

Figure 14. Pitching and plunging wing: number of incremental steps r_n as a function of the time step number n. Solid line: ball-vertex method; dashed line: torsional spring method.

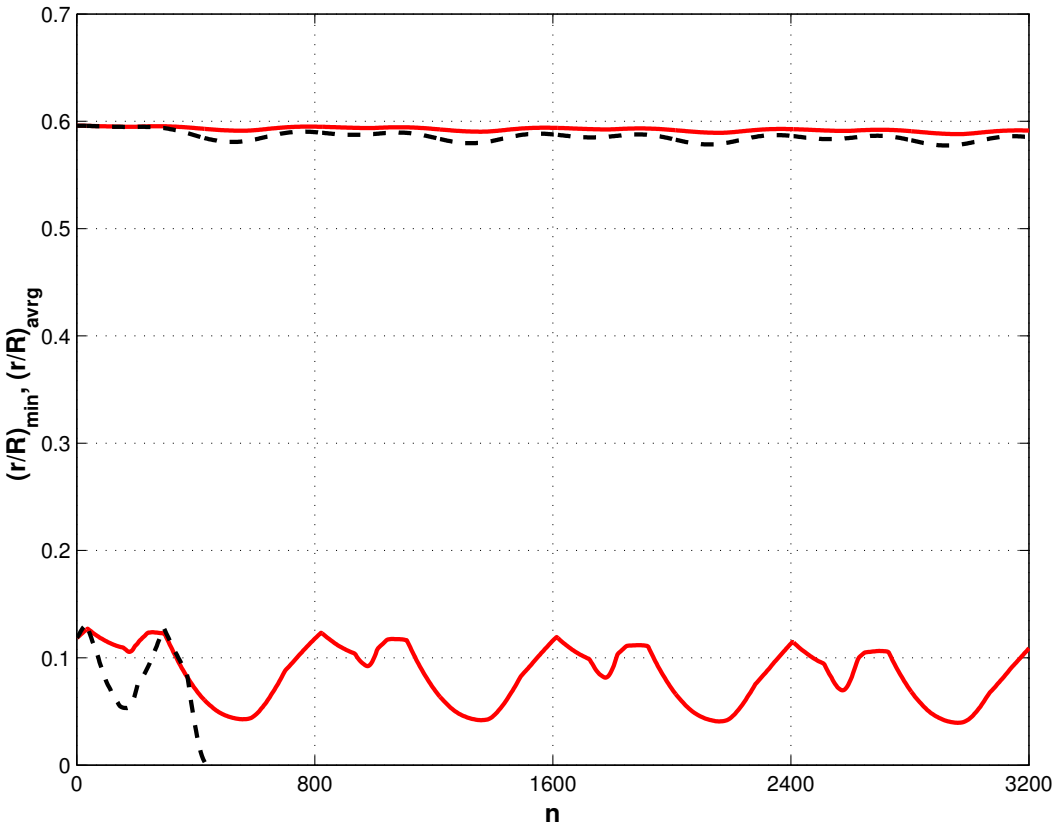

Figure 15. Pitching and plunging wing: minimum, $(r/R)_{\min}$, and average, $(r/R)_{\text{avrg}}$, radii ratios as functions of the time step number n. Solid line: ball-vertex method; dashed line: torsional spring method.

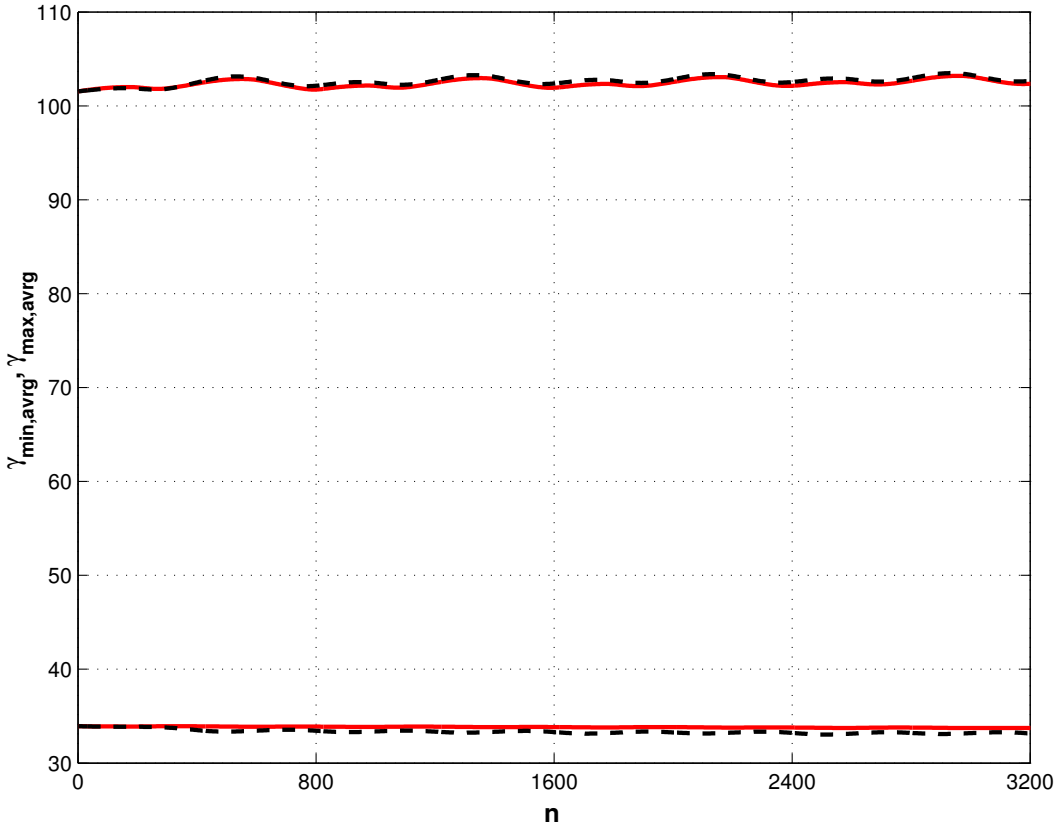

Figure 16. Pitching and plunging wing: average minimum, $\gamma_{\mathrm{min,avrg}}$, and average maximum, $\gamma_{\mathrm{max,avrg}}$, dihedral angles as functions of the time step number n. Solid line: ball-vertex method; dashed line: torsional spring method.

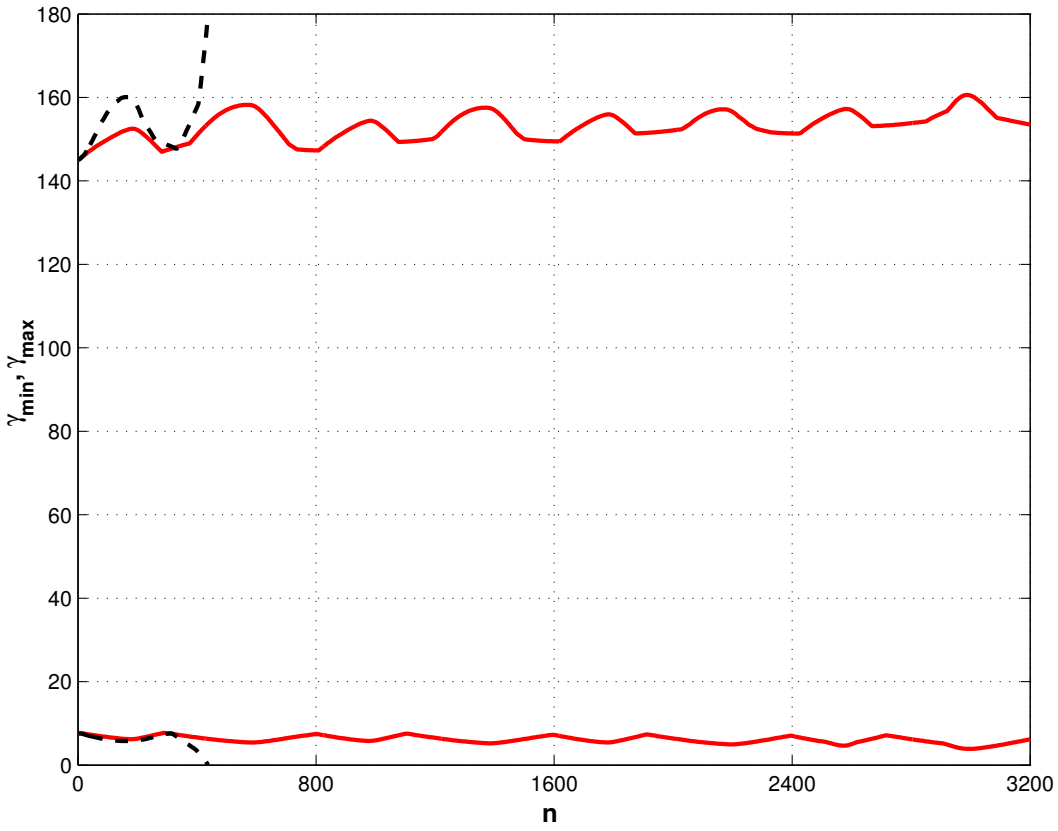

Figure 17. Pitching and plunging wing: worst minimum, $\gamma_{\min}$, and worst maximum, $\gamma_{\max}$, dihedral angles as functions of the time step number n. Solid line: ball-vertex method; dashed line: torsional spring method.

input : $\mathcal{T}_h, \Gamma_{m,h}, \boldsymbol{g}_i \; \forall \, i \in \{V\}_{\Gamma_{m,h}}, r_{\max}$

Compute spring stiffnesses at current configuration $\boldsymbol{x}_i \; \forall \, i \in \{V\}_{\mathcal{T}_h}$
$\boldsymbol{x}_i^1 = \boldsymbol{x}_i \; \forall \, i \in \{V\}_{\mathcal{T}_h}$
$r = 0$
do
  $r = r + 1$
  Compute increment coefficient α^r at $\boldsymbol{x}_i^r \; \forall \, i \in \{V\}_{\mathcal{T}_h}$
  $\boldsymbol{u}_i^r = \alpha^r \boldsymbol{g}_i \; \forall \, i \in \{V\}_{\Gamma_{m,h}}$ *[Current imposed displacements]*
  Solve linear problem and compute vertex displacements $\boldsymbol{u}_i^r \; \forall \, i \in \{V\}_{\mathcal{T}_h - \Gamma_{m,h} - \Gamma_{0,h}}$
  $\boldsymbol{x}_i^{r+1} = \boldsymbol{x}_i^1 + \boldsymbol{u}_i^r \; \forall \, i \in \{V\}_{\mathcal{T}_h - \Gamma_{0,h}}$ *[Update vertex positions]*
while $((\alpha^r < 1)$ **and** $(r \leq r_{\max}))$
$\boldsymbol{x}_i^{r+1} = \mathcal{P}(T, \boldsymbol{x}_i^{r+1}) \; \forall \, i \in \{V\}_{\Gamma_{s,h}}$ *[Project vertex position onto model entity T, if T is a face or edge of Γ_s]*

Figure 18. Incremental displacement algorithm.

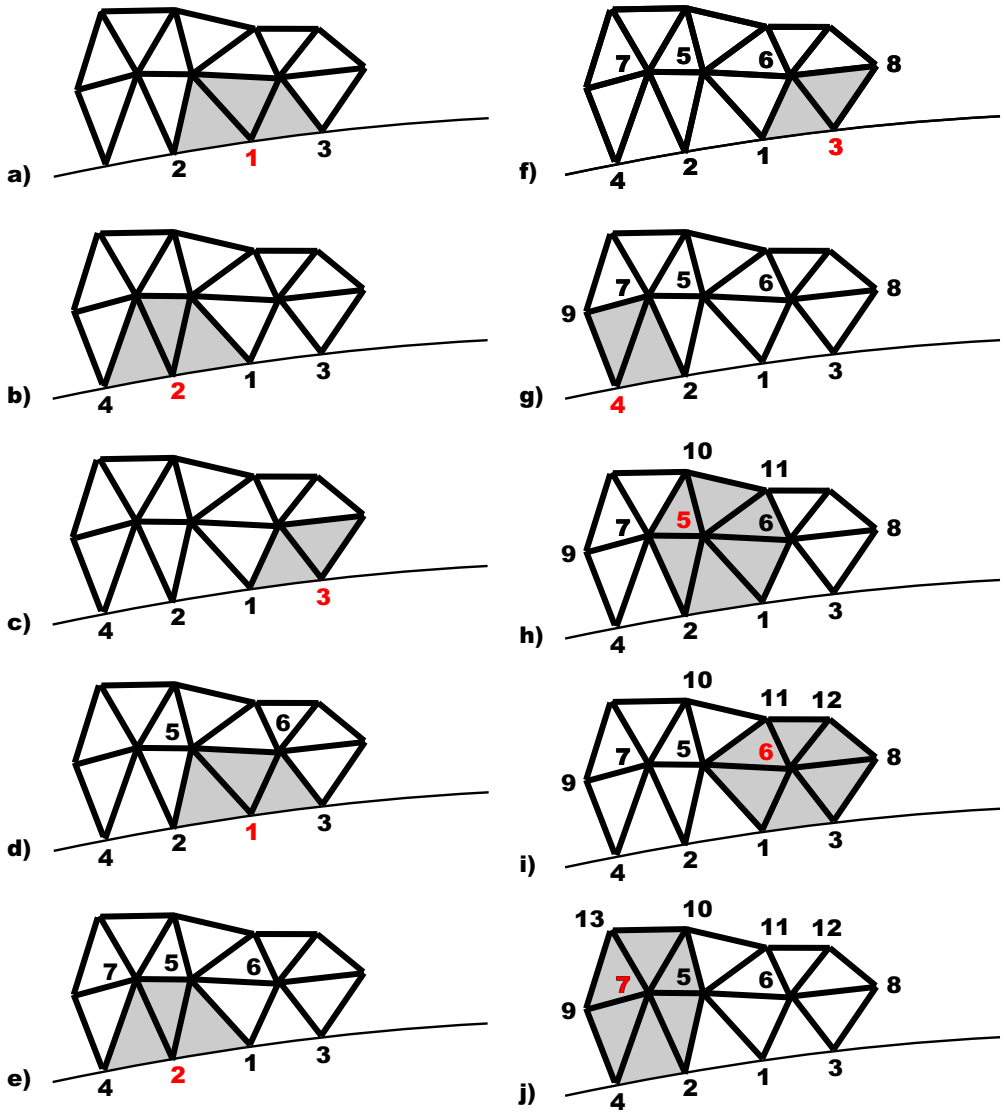

Figure 19. Construction of a line ordering in an unstructured mesh using the "onion leaves" greedy algorithm.