Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computer Physics Communications 172 (2005) 86–108
Computer Physics Communications
www.elsevier.com/locate/cpc

Parallel dynamic and kinetic regular triangulation in three dimensions

Tilo Beyer [a,b,d,e,*], Gernot Schaller [b], Andreas Deutsch [a], Michael Meyer-Hermann [c,e]

[a] *Centre for Information Technology Services and High Performance Computing, Dresden University of Technology, Zellescher Weg 12, 01062 Dresden, Germany*
[b] *Institute for Theoretical Physics, Dresden University of Technology, Zellescher Weg 17, 01062 Dresden, Germany*
[c] *Centre for Mathematical Biology, Mathematical Institute, Oxford University, 24-29 St Giles', Oxford OX1 3LB, UK*
[d] *Interdisciplinary Centre for Bioinformatics, University of Leipzig, Härtelstr. 16-18, 04107 Leipzig, Germany*
[e] *FIAS, University of Frankfurt, Max-von-Laue-Str. 1, 60438 Frankfurt a.M., Germany*

Received 29 January 2005; received in revised form 10 June 2005; accepted 26 June 2005

Available online 18 August 2005

Abstract

A parallel algorithm for regular triangulations is presented. For the purpose of fully dynamic and kinetic particle simulations it allows vertex insertion, deletion, movement, and weight changes. We describe new algorithms for incremental construction of regular triangulations, parallel vertex deletion and insertion. Finally, a parallel Lawson flip algorithm for vertex displacements is presented. The performance analysis demonstrates a significant parallel efficiency for various system sizes and performed changes.
© 2005 Elsevier B.V. All rights reserved.

PACS: 87.18.-h; 87.18.Bb; 87.18.Hf; 81.05.Rm

Keywords: Delaunay triangulation; Voronoi tessellation; Data-parallelism; Task-parallelism; Particle simulations

1. Introduction

Many simulations in science are based on discrete objects under the influence of short-ranged or contact-dependent interactions [1]. The influence of such objects or particles on each other can be modeled by next neighbor relationships. In such applications, the number of interactions is approximately linear in the number of particles. This favors the use of a linear sized graph to describe the possible two-body interactions. In the present article the Delaunay—or more generally the regular—triangulation [2] is considered for defining neighborship relations. We focus on the use of regular triangulations in order to account for the influence of various particles sizes on the neighborhood topology. The type of model that is supported by the presented algorithms are three-dimensional individual-based simulations of spherical particles of various size

* Corresponding author. Tel.: +49 69 798 47500; Fax: +49 69 798 47611.
E-mail address: t.beyer@figss.uni-frankfurt.de (T. Beyer).

0010-4655/$ – see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.cpc.2005.06.009

with next neighbor interactions. An example are granular media [3]. In addition the incremental insertion or removal of particles is possible which is suitable for our target application: The modeling of biological tissues with single-cell-based off-lattice simulations in three dimensions.

Various single-cell-based spatial simulations of biological tissues are calculated on lattices, e.g., [4,5]. Only a few are based on off-lattice techniques [6–9]. A major problem of the latter is the required computer power caused by additional degrees of freedom regarding the neighborhood topology and the position of particles, even when using the linear sized regular triangulations. Current algorithms applied to the problem of regular triangulations are mostly dynamic, allowing only the insertion and deletion of vertices [10, 11]. From the point of view of tissue simulation this permits only the description of birth and death of cells while the movement of cells requires a re-triangulation of the whole set of vertices. For comparable problems in the physics of granular media the regular triangulation approaches is used to describe the neighborship relations of moving spheres [3]. The implementations are limited to small numbers of particles and a slowly changing neighborhood, i.e. in these applications the triangulation has to be recomputed only for a limited number of times. Both limitations need to be dropped for a realistic description of biological tissues as tumor spheroids [9] or immunological tissue [12].

To overcome current computer hardware limits it is desirable to develop parallel algorithms for dynamic (vertex insertion and removal) and kinetic (vertex displacement) regular triangulations. Although serial algorithms exist to either dynamic or kinetic vertices [3,10,11,13–18], parallel solutions in three dimensions are limited to solutions of subproblems. To our knowledge no parallel deletion algorithms exist while vertex insertion is restricted to the Delaunay triangulation, i.e. not covering the more general regular triangulation [19,20]. The only parallel solution for kinetic vertices is based on shared memory architectures and limited to small displacements [3]. The task to generate an initial Delaunay triangulation is achieved by many parallel solutions [21–25], but none of these can generate regular triangulations. Additionally a part of these solutions relies on specialized architectures like shared memory limiting portability of the algorithm and code.

This work presents a parallel algorithm for the construction and maintenance of regular triangulations supporting dynamic vertices (insertion and deletion) as well as a parallel scheme for vertex kinetics (movement). The tool is appropriate for arbitrary partitions of vertices and for arbitrary vertex distributions. An important issue of the present work is to avoid a predetermined partition of the vertices like in the projection-based algorithms [23,24]. This allows to distribute vertices according to the needs of the application rather than to an optimal partition for the triangulation itself.

We use data-parallelism as an underlying parallelization strategy in combination with task-parallel execution of algorithms. The initial construction is computed by extending the incremental construction method [21,22] to regular triangulations. The parallel deletion routine is based on a task-parallel variant of the incremental construction method. The dynamic insertion of vertices relies on a concurrent Bowyer–Watson algorithm [14,15,19]. Based on previous work of our group [18] we use the Lawson flip algorithm [13] for kinetic vertices in conjunction with a new solution to parallelize it. Performance experiments show that we achieve reasonable scaling with the number of processors used for the dynamic and kinetic algorithms even for small numbers of vertices.

2. Terms and definitions

We first define the regular triangulation and related geometric objects. Then we introduce the terminology of parallel computing as it is used here. All algorithms are presented for three-dimensional space though for clarity figures illustrate the two-dimensional case if not stated otherwise.

2.1. Regular triangulations

$\mathbf{p} \in \mathbb{R}^3$ refers to a *point* in space. A *vertex* $P = (\mathbf{p}, w_p) \in \mathbb{R}^3 \times \mathbb{R}$ denotes a point $\mathbf{p}$ in three-dimensional space with an associated positive weight w_p.[1]

A *simplex* $\sigma_{\langle A,B,C,D\rangle}$ is the convex hull of four non-coplanar vertices A, B, C, D. These will be called the

[1] We do not consider the formally possible case of negative weights.

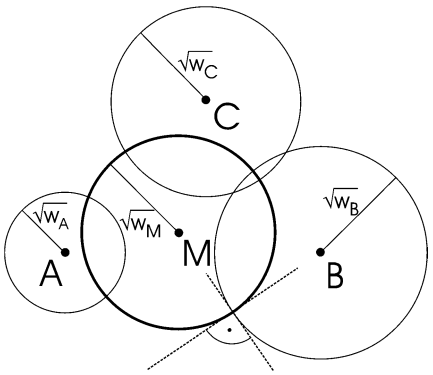

Fig. 1. The orthocircle M (bold outlined circle) of the simplex $\sigma_{\langle A,B,C\rangle}$ perpendicularly intersects all circles associated with the vertices A, B, C. The weight of each vertex is represented as the radius $\sqrt{w}$ of the circles.

endpoints of the simplex. Then a *triangulation* $\mathcal{T}$ of a set of vertices $\mathcal{V}$ is a collection of non-overlapping simplices covering the convex hull of $\mathcal{V}$ such that every vertex of every simplex $\sigma \in \mathcal{T}$ is element of the point set $\mathcal{V}$.

We define the *power distance* $p(P, Q)$ between the vertices $P = (\mathbf{p}, w_p)$ and $Q = (\mathbf{q}, w_q)$ to be:

$$p(P, Q) = \|\mathbf{p} - \mathbf{q}\|^2 - w_p - w_q \tag{1}$$

with $\|.\|$ denoting the Euclidean norm in $\mathbb{R}^3$.

The *orthosphere* of a simplex $\sigma_{\langle A,B,C,D\rangle}$ with center $\mathbf{m}$ and radius $\sqrt{w_m}$ is defined such that with $M = (\mathbf{m}, w_m)$

$$p(V, M) = 0, \tag{2}$$

holds for every vertex $V \in \{A, B, C, D\}$. If all w_v and w_m are positive then the orthosphere can be interpreted as a sphere of radius $\sqrt{w_m}$ with center $\mathbf{m}$ which intersects each sphere with radius $\sqrt{w_v}$ around the points $\mathbf{v} = \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$ orthogonally (Fig. 1). The orthosphere M is said to *contain* a vertex P if $p(P, M) < 0$.

The triangulation $\mathcal{T}$ is said to be a *regular triangulation* (also known as weighted Delaunay triangulation) if the orthosphere of every simplex $\sigma_{\langle A,B,C,D\rangle} \in \mathcal{T}$ contains no vertex. The dual graph of the regular triangulation is known as generalized Voronoi or power diagram which always exists [1,26] and guarantees the existence of the regular triangulation (for more properties of regular triangulations see [10]). The well-known relation between regular triangulations in dimension d and convex hulls in $d + 1$ [27] leads to a simple criterion to check whether the vertex $V = (\mathbf{v}, w_v)$ is contained in the orthosphere of a simplex constructed with the vertices $A = (\mathbf{a}, w_a)$, $B = (\mathbf{b}, w_b)$, $C = (\mathbf{c}, w_c)$, $D = (\mathbf{d}, w_d)$. If the vertices are positively oriented, i.e. if

$$\mathrm{O}(A, B, C, D) \stackrel{\text{def}}{=} \begin{vmatrix} a_x & a_y & a_z & 1 \\ b_x & b_y & b_z & 1 \\ c_x & c_y & c_z & 1 \\ d_x & d_y & d_z & 1 \end{vmatrix} > 0, \tag{3}$$

then V is contained within the orthosphere if

$$\mathrm{S}(A, B, C, D, V) \stackrel{\text{def}}{=} \begin{vmatrix} a_x - v_x & a_y - v_y & a_z - v_z & \|\mathbf{a} - \mathbf{v}\|^2 - w_a + w_v \\ b_x - v_x & b_y - v_y & b_z - v_z & \|\mathbf{b} - \mathbf{v}\|^2 - w_b + w_v \\ c_x - v_x & c_y - v_y & c_z - v_z & \|\mathbf{c} - \mathbf{v}\|^2 - w_c + w_v \\ d_x - v_x & d_y - v_y & d_z - v_z & \|\mathbf{d} - \mathbf{v}\|^2 - w_d + w_v \end{vmatrix} > 0 \tag{4}$$

holds and V is said to violate the *Delaunay criterion* of simplex $\sigma_{\langle A,B,C,D\rangle}$. The simplex is called *invalidated by vertex* V or *non-regular*. A triangulation is called non-regular if any of its simplices is non-regular.

We call the points of a given point set $\mathcal{V}$ to be in *general position* [28] if

- no four points are coplanar, and
- no five points are co(ortho)spherical.

2.2. Elementary topological transformations

Many of the parallel algorithms rely on Lawson's flip algorithm [13] which has been extended to three dimensions [16,17] and regular triangulations [10] before. We will briefly describe the algorithm using the general position assumption.

It has been shown that under certain circumstances [10] a non-regular triangulation can be transformed into a regular one using sequences of the following local topological transformations (also known as *flips*) of sets of five points. The first flip of type $1 \to 4$ replaces one simplex and a vertex inside the simplex by four simplices. The second flip $4 \to 1$ reverses this transformation. The second pair consists of the flip $2 \to 3$ which replaces two simplices by three, and the flip $3 \to 2$ reversing this transformation (Fig. 2). For the last two transformations all vertices have to be in a convex configuration. A flip is performed when the Delaunay criterion $\mathrm{S}(A, B, C, D, V)$ (Eq. (4)) of one of the simplices $\sigma_{\langle A,B,C,D\rangle}$ is violated by the fifth vertex V.

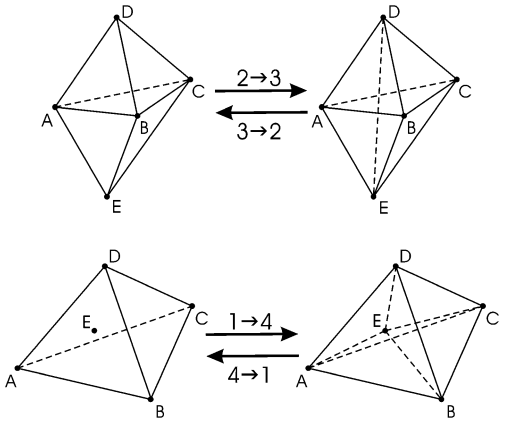

Fig. 2. The four possible flips in three dimensions are shown. Invisible edges have been drawn by dashed lines. The upper panel shows the replacement of a triangle $\langle A, B, C\rangle$ by an edge $\langle D, E\rangle$ (flip $2 \to 3$). The reverse operation is the flip $3 \to 2$. All five vertices have to be in a convex configuration. The lower panel shows the flip $1 \to 4$ inserting vertex E into the simplex $\langle A, B, C, D\rangle$ replacing it by four simplices. The reverse operation flip $4 \to 1$ removes a vertex from a triangulation. In that case the vertex E has to be inside the convex hull (simplex) of the vertices A, B, C, D.

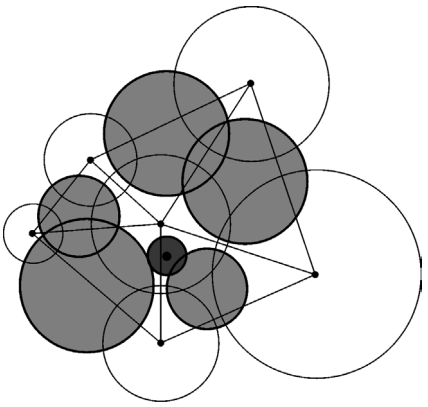

Fig. 3. Illustration of redundant vertices. A vertex is not inserted into the triangulation when it does not invalidate any simplex. The positive weights of the triangulated vertices are shown as empty circles. The redundant vertex (with the dark grey circle representing its weight) is, according to the power distance Eq. (1), not contained in the orthocircles of the simplices (light grey circles).

If a flip $4 \to 1$ is performed then a vertex which is not endpoint of any simplex, is created. This vertex is called a *redundant vertex* [10,11] (Fig. 3). A redundant vertex will not be re-inserted via flip $1 \to 4$ as long as no other vertex changes its position, weight, or is removed from the triangulation [11].

2.3. Parallel terminology and general parallelization scheme

The aim of this work is to generate a portable parallel program. This leads us to use the Message Passing Interface (MPI) [29]. In this context a *process* is a copy of a program running on some computer communicating with other processes on the same or other computers. The communication functions are defined in the MPI standard. Each of these processes is assigned a unique identifier called *rank*.

The paradigm used for parallelization is data-parallelism. To allow communication between processes we need a global data structure with subsets of the data locally stored at each process. For that purpose we uniquely attribute every vertex to some process p_i. We call this vertex *owned* by the process p_i. Globally, a vertex V can be identified by a pair of indices $V \mapsto [p_V, n_V]$ where p_V and n_V are the process owning it and a locally unique index, respectively. This ensures that processes do not need to communicate to create globally unique indices. Each index pair is coupled with a pointer to the data structure of the vertex. We call that index pair together with the pointer a *metapointer* [3].

The basic data structure for the regular triangulation is a simplex list. Each simplex consists of four metapointers to its endpoints. Additionally, every simplex is associated itself with a metapointer. Simplices are connected amongst each other via their simplex metapointer which is useful within the flip algorithm. The difference to the metapointers of vertices is that the first index does not imply that the corresponding process owns this simplex—it is just a unique identifier. Simplices can be shared amongst processes depending on which of its endpoints belongs to which process. Consequently, every process needs a copy of vertices of its *neighbor processes* to which the vertex metapointers of the simplex point to. A *neighbor process* of a process p_i is defined as every process p_j with which p_i shares at least one simplex. We will refer to a shared simplex as *boundary simplex*. They mark the connection of parts of the triangulations associated with each process (Fig. 4).

For simplicity if the common face of two simplices involves only vertices owned by neighbor processes the neighborship relation is not set (Fig. 4). Thus, it is determined which simplices are stored at which

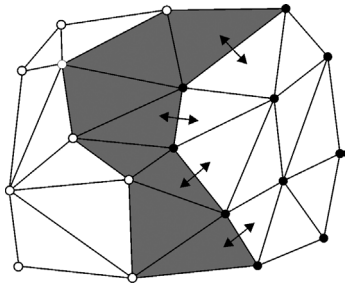

Fig. 4. A two-dimensional example of data-parallelism. Two sets of vertices (open and closed circles) are assigned to two distinct processes. The white simplices (triangles) exist only once at each process while the grey-shaded ones are boundary simplices and present at both processes. The shaded simplices have a unique metapointer despite the fact that they are stored at both processes. The double-headed arrows indicate which inter-simplex links are not visible for the process associated with the open circles. These links are only visible for the second process (closed circles).

process thereby defining the *domain* that a process is working on.

The concept of metapointers allows to identify vertices and simplices when exchanging information between processes and allows easy access to geometrically identical objects stored at different processes.

In addition, direct vertex–vertex connections are part of the global vertex data structure in order to facilitate access to this information important in many applications.

Essentially all algorithms provided here modify a collection of simplices by replacement of a part of the triangulation with a different triangulation. Within the parallel algorithms however, the modification of boundary simplices is challenging as the processes may perform different changes to the triangulation which can interfere. Therefore it is required to guarantee a unique solution of the algorithm by allowing only one process to operate on a given subset of the triangulation. Then the result is sent to the neighbor processes sharing the simplices. For this purpose we introduce the term *access rights*: The *blocking status* of a simplex guarantees that only one process p_i has access rights. We call such a simplex blocked by process p_i. This requires communication with neighbor processes to provide unique access rights amongst the participating processes.

Access restriction does not guarantee that successive changes to the triangulation performed by one process are performed in the same order at a neighbor process: Assume process p_1 shares a part of the triangulation with processes p_2 and p_3. Process p_1 induces changes of the triangulation followed by process p_2. Each of the processes p_1 and p_2 send requests to process p_3 for an update of the shared part of the triangulation. These messages may be received in reversed order by the process p_3 as the MPI standard guarantees only correct message ordering between two processes. The solution of this *message delay* problem is given explicitly for every algorithm. Note, that for vertices the access rights are defined *a priori* by their metapointer and remain unchanged. In contrast simplex access rights are a dynamical property within the different algorithms.

3. Initial triangulation

The initial triangulation of a point set takes advantage of the incremental construction of Delaunay triangulations [30,31]. The unweighted case was already parallelized previously [21,22]. We will expand it to regular triangulations. The presented algorithm works efficiently for uniformly distributed point sets and small vertex weights w_v, i.e. if

$$\sqrt{|w_v|} \lesssim l$$

holds with $l = (V_{\text{cube}}/N_{\text{vert}})^{1/3}$ being the average distance between N_{vert} uniformly distributed vertices inside a cube of volume V_{cube}. In that regime the triangulation is constructed in approximately linear time.

3.1. Recall of incremental construction

The incremental construction algorithm starts from an initial simplex that fulfills the Delaunay criterion. The algorithm inserts all faces of that simplex in a so-called active face list (AFL) [21]. Every face in the AFL will be expanded to a Delaunay simplex. This is done by searching through all vertices until the new simplex fulfills the Delaunay criterion. Naturally, the search considers only vertices in the half space opposite to the simplex generating the face. When a new simplex is constructed it generates three new faces. If any of those faces is already in the AFL it will be removed completely from the AFL—otherwise it will be added for expansion. If no vertex can be found to expand a given face into a simplex then the face belongs

to the boundary of the convex hull and will be removed from the AFL as well. This algorithm is expensive as it is of complexity $\mathcal{O}(N^3)$ for N vertices.

In practical implementations however one can achieve an almost linear performance by several improvements: First, the check if a new face is present in the AFL can be done using hash maps resulting in an amortized constant time operation [21]. A second improvement is to use a bucketing technique for localization of vertices that might result in a Delaunay simplex [32]. A simple method are uniform grids [21]. One assigns every vertex to a grid cell and performs bounding box or spiral searches on the grid starting close to the face to expand. For uniform vertex distributions a vertex is found in constant time. Other distributions may take longer and other grid techniques might be more useful. Note, however, that this technique has been slightly modified (see Section 5) in order to conveniently treat kinetic vertices.

Another improvement is to simplify the computation of the Delaunay criterion. To prevent the algorithm to evaluate the whole determinant (Eq. (4)) for every vertex one can use the signed Delaunay distance SD of a point $\mathbf{d} \in \mathbb{R}^3$ from a face $\langle \mathbf{a}, \mathbf{b}, \mathbf{c} \rangle$ [21,22],

$$\mathrm{SD}(\mathbf{d}) \stackrel{\text{def}}{=} \left(\mathbf{m}(\mathbf{d}) - \mathbf{m}_{\langle \mathbf{a},\mathbf{b},\mathbf{c}\rangle}\right) \cdot \frac{(\mathbf{a}-\mathbf{b}) \times (\mathbf{a}-\mathbf{c})}{\|(\mathbf{a}-\mathbf{b}) \times (\mathbf{a}-\mathbf{c})\|}, \tag{5}$$

with $\mathbf{m}_{\langle \mathbf{a},\mathbf{b},\mathbf{c}\rangle}$ the center of the circumcircle of the face $\langle \mathbf{a}, \mathbf{b}, \mathbf{c} \rangle$ and $\mathbf{m}(\mathbf{d})$ the center of the circumsphere formed by the four points $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, $\mathbf{d}$ which depends only on the point $\mathbf{d}$ for the given face $\langle \mathbf{a}, \mathbf{b}, \mathbf{c} \rangle$. It can be shown that finding the vertex that forms a Delaunay simplex with the face $\langle \mathbf{a}, \mathbf{b}, \mathbf{c} \rangle$ is equivalent to finding a vertex that minimizes $\mathrm{SD}(\mathbf{d})$ [22]. With some transformations (Appendix A) one can further reduce this equation to compute only terms depending on the point $\mathbf{d}$ [22]. This significantly reduces the number of necessary computations.

In order to construct the initial simplex first a Delaunay edge has to be found. A sufficient condition for an edge fulfilling the Delaunay criterion is that the circumsphere with minimal radius contains no other vertex. One possibility is to search for the vertex closest to a given starting vertex. This edge can be expanded to a Delaunay face by minimizing a two-dimensional version of the signed Delaunay distance. Then this face is expanded to the initial simplex as described above but without restricting the vertex search to a single half space.

3.2. Parallel incremental construction for unweighted Delaunay triangulations

The parallelization of the above algorithm for incremental Delaunay construction is straightforward (Fig. 6, left panel). In contrast to previous work we do not use a geometric partition in the sense of a merge-first divide and conquer algorithm [21]. We assign the vertices to the processes using a space filling curve. For this purpose we use the partition functions from the ParMetis library [33]. In this procedure every vertex is uniquely assigned to a process but every process holds a temporary copy of the whole set. Every process p_i triangulates its vertices stopping at its domain boundary. The boundary is defined by the Delaunay faces consisting only of vertices belonging to other processes $p_{(j\neq i)}$.

It can happen that the assignment of vertices to processes results in non-simply connected domains of the triangulation as the partition function does not take care of that. Therefore every vertex attributed to a process is checked if it is vertex of at least one simplex. If a vertex is found that is not connected it is used to construct another initial simplex which then is expanded to another domain until all vertices are connected to simplices. Note, that this is a procedure for security and a rare event in most situations.

After the domain of process p_i has been triangulated the process will contact its neighbor processes. The corresponding contacts are automatically generated during the construction of the boundary faces. Neighbor processes are contacted to link together their common simplices assigning a unique metapointer to each simplex (Fig. 4). In this way the initial process communication topology is defined which will be used in the other algorithms. The temporary copies of vertices from unconnected processes are removed and only those of neighbor processes remain.

3.3. Extension to regular triangulations

The extension of the signed Delaunay distance (Eq. (5)) to regular triangulations is straightforward by replacing the points by vertices with $\mathbf{m}(D)$ and

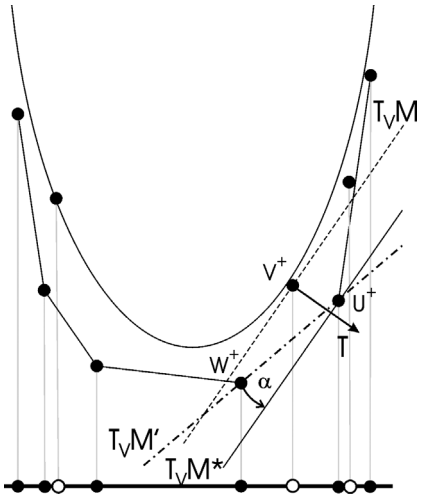

Fig. 5. The two-dimensional lifted map of a one-dimensional regular triangulation demonstrating how to find a starting vertex for the incremental construction. The paraboloid of a standard Delaunay triangulation is shown as reference. The vertices are shifted downwards (assuming positive weights). Starting the algorithm with the (lifted) vertex V^+ one finds the tangential plane $T_V M$ (dashed line) with normal vector T. The tangential plane $T_V M$ is shifted along T until no vertex is on its positive side (defined by the normal vector T) thus finding the vertex U^+ as start vertex and the shifted plane $T_V M^\star$ (full line). To get the initial simplex (edge in this example) one searches for the vertex W^+ which minimizes the angle α between the shifted plane $T_V M^\star$ and the line connecting the start vertex U^+ with itself resulting in the plane $T_V M'$ (dashed dotted line). Open circles denote redundant vertices of the triangulation.

$\mathbf{m}_{\langle A,B,C\rangle}$ now being the centers of the orthospheres (see also Appendix A). The major change to the original algorithm is the detection of redundant vertices. For example, the simplex algorithm used to determine the initial simplex for Delaunay triangulations will fail if one accidentally starts with a redundant vertex. This can be avoided by consideration of convex hulls: The Delaunay triangulation is related to the convex hull [27] by the lifted map $\mathbf{p} \in \mathbb{R}^3 \mapsto p^+ = (\mathbf{p}, \|\mathbf{p}\|^2) = (p_x, p_y, p_z, \|\mathbf{p}\|^2) \in \mathbb{R}^4$ (Fig. 5). This translates the incremental Delaunay construction into a gift-wrapping algorithm for convex hulls. The corresponding lifted map $\mathcal{V}^+$ of a set of weighted vertices $\mathcal{V}$ is $V \in \mathcal{V} \subset \mathbb{R}^3 \times \mathbb{R} \mapsto V^+ = (\mathbf{v}, \|\mathbf{v}\|^2 - w_v) = (v_x, v_y, v_z, \|\mathbf{v}\|^2 - w_v) \in \mathbb{R}^4$. In the case of equal weights all vertices lie on a paraboloid in $\mathbb{R}^4$. With different weights w_v lifted vertices V^+ are shifted below or above (in the case of negative weights) the paraboloid.

The starting simplex is found as follows (Fig. 6, right panel):

- The tangential hyper plane $T_V M$ of the paraboloid in $\mathbb{R}^4$ generated by the lifting map is calculated for an arbitrary vertex V. The tangential plane through the point $V^+ = (\mathbf{v}, \|\mathbf{v}\|^2 - w_v) \in \mathbb{R}^4$ is given by its normal T

$$T = \frac{1}{\sqrt{4\|\mathbf{v}\|^2 + 1}}(2\mathbf{v}, -1)\,. \tag{6}$$

  If no lifted vertex D^+ lies on the positive side of the tangential hyper plane $T_V M$, i.e. if

$$T \cdot (D^+ - V^+) = \mathbf{t} \cdot (\mathbf{d} - \mathbf{v}) - \left(\|\mathbf{d}\|^2 - \|\mathbf{v}\|^2 - w_d + w_v\right) < 0 \quad \forall D^+ \in \mathcal{V}^+, \tag{7}$$

  then the vertex is not redundant. Note, that $T \cdot (D^+ - V^+) > 0$ does not necessarily mean that the vertex is redundant. In the case of redundancy the tangential hyper plane $T_V M$ is parallel shifted into the point D^+ and the search is repeated until a proper vertex U^+ and the shifted hyperplane $T_V M^*$ is found (Fig. 5).
- From U^+ the initial edge is computed by searching for the vertex W^+ which minimizes the angle α:

$$\cos\alpha = \frac{T \cdot (W^+ - U^+)}{\|W^+ - U^+\|_4} \tag{8}$$

  (with $\|.\|_4$ being the Euclidean norm in $\mathbb{R}^4$). In $\mathbb{R}^3$ the initial edge is then given by $\langle U, W\rangle$.
- To find the first face of the initial simplex one 'tilts' the tangential hyper plane $T_V M$ into a hyper plane $T'_V M$ containing the edge so that the normal T' becomes

$$T' = \frac{N}{\|N\|_4} \quad \text{with} \quad N = T - (W^+ - U^+)\frac{T \cdot (W^+ - U^+)}{\|W^+ - U^+\|_4}\,. \tag{9}$$

  Then the first face is constructed with the vertex Y^+ that minimizes the angle β between the hyper plane $T'_V M$ and the orthogonal projection P of the vector $Y^+ - U^+$ with respect to the edge formed

```
begin incremental_construction(V)
create uniform grid
call find_initial_simplex()
while AFL not empty do
  face f ← f ∈ AFL
  for all vertices D ∈ V do
    if D in positive half space of f and
      SD(D) < SD(V) then
      vertex V ← D
    end if
  end for
  if V not empty then
    create simplex σ⟨A,B,C,V⟩
    for all nf ∈ new faces do
      if nf ∉ AFL then
        if nf ≠ boundary face then
          AFL ← nf
        end if
      else
        remove nf from AFL
      end if
    end for
  else
    border of domain
  end if
  remove face ⟨A, B, C⟩ from AFL
end while
synchronize simplices
end incremental_construction
```

```
begin find_initial_simplex()
calculate T_V M from V⁺ ∈ V⁺
for all D⁺ ∈ V⁺, D⁺ ≠ V⁺ do
  if T · (D⁺ − V⁺) > 0 then
    U⁺ ← D⁺
  end if
end for
for all D⁺ ∈ V⁺, D⁺ ≠ U⁺ do
  if T·(D⁺−U⁺)/‖D⁺−U⁺‖₄ < T·(W⁺−U⁺)/‖W⁺−U⁺‖₄ then
    W⁺ ← D⁺
  end if
end for
T ← T − (W⁺ − U⁺) T·(W⁺−U⁺)/‖W⁺−U⁺‖₄
T ← T/‖T‖₄
Q ← W⁺ − U⁺
Q ← Q/‖Q‖₄
for all D⁺ ∈ V⁺, D⁺ ≠ U⁺, W⁺ do
  P ← D⁺ − U⁺ − Q[(D⁺ − U⁺) · Q]
  P′ ← Y⁺ − U⁺ − Q[(Y⁺ − U⁺) · Q]
  if T·P/‖P‖₄ < T·P′/‖P′‖₄ then
    Y⁺ ← D⁺
  end if
end for
for all D ∈ V do
  if SD(D) < SD(X) then
    X ← D
  end if
end for
create simplex σ⟨U,W,Y,X⟩
end find_initial_simplex
```

Fig. 6. Pseudo-code of initial triangulation of a set of vertices $\mathcal{V}$. Only the creation of the uniform grid and the synchronization of simplices requires communication. The vertex searches are optimized using the uniform grid. See text for symbols.

by U^+ and W^+:

$$\cos\beta = \frac{T' \cdot P}{\|P\|_4} \quad \text{with}$$
$$P = Y^+ - U^+ - (W^+ - U^+) \times \frac{(Y^+ - U^+) \cdot (W^+ - U^+)}{\|W^+ - U^+\|_4^2}. \tag{10}$$

- To complete the initial simplex $\sigma_{\langle U,W,Y,X\rangle}$ the signed Delaunay distance to the constructed face $\langle U, W, Y\rangle$ is minimized.

The full triangulation is constructed as before by minimizing the signed Delaunay distance. Any vertex which is left without an incident simplex is a redundant vertex and will get marked as such.

Parallelization of this scheme implies several problems: The initial simplex found by the algorithm may not contain any of the vertices associated with the process calculating it, i.e. U, W, Y, X belong to neighbor processes despite that the starting vertex V belongs to the process. In addition the identification of redundant vertices is non-trivial. In contrast to the serial code, vertices which are not vertex of a simplex are not necessarily redundant but may be part of a simplex that belongs to an unconnected part of the triangulated area. This is related to non-simply connected domains that may emerge by a bad partition of

the vertices. We solved these problems with an algorithm that we termed *construction hopping*. In analogy to the simplex visibility walk during incremental insertion algorithms [18,34] one chooses a face of the found initial simplex that is oriented towards the desired initial vertex. This face is expanded as in the normal construction process. Again one of the faces of the newly constructed simplex is chosen and expanded. This procedure is repeated until one of the following three cases occurs: (i) A simplex that contains the start vertex is found and can be used as initial simplex. (ii) Another vertex owned by the process is vertex of a simplex which now serves as initial simplex. (iii) The chosen vertex is redundant and lies within a simplex constructed during construction hopping. Then another vertex is used to start the initial simplex search from the beginning. The identification of redundant vertices is done using either the simplex visibility walk or—if it fails—by construction hopping.

The vertex search on the grid requires a modification of the bucketing technique as the weight of vertices alters the minimization of the signed Delaunay distance. We interpret the weight as a sphere mapped onto the grid eventually covering more than one grid cell. As this work focuses on tissue modeling the spheres (thought of as cells) are of similar diameter as the minimal distance between vertices. Thus every sphere is likely to cover only a small part of the uniform grid. For significantly larger weights w_v the search on the uniform grid becomes inefficient as a large fraction of all vertices is attributed to a single grid cell. This vertex search algorithm scales linearly with the numbers of vertices thus resulting in an overall quadratic construction algorithm.

3.4. Robustness issues

All calculations are performed with double precision floating point numbers. Generally, using floating point representations of positions and topological properties suffers from numerical imprecision of computer calculations. To deal with the rare cases of inconsistent triangulations (less than 1% of the tested uniform distributions) two approaches are used. The first one is an *a posteriori* error correction algorithm which relies on the distortion of the input vertices [28]. After the distortion the whole triangulation is recomputed from scratch. This also ensures that the vertices are in general position (Section 2.1) which guarantees a unique solution for the triangulation. The second approach uses interval arithmetics [35] provided by the BOOST library [36] to filter critical cases. If this indicates critical numerical errors of floating point calculations the orientation or orthosphere primitive is calculated with adaptive precision instead of computing the signed Delaunay distance. The hand-tuned adaptive precision primitives of Shewchuk [37] are used. For our purpose we extended the circumsphere primitive to the orthosphere primitive. It is up to the user to decide if the *a posteriori* vertex distortion or error detection by interval arithmetics fits best to the desired application.

Note, if the construction of the initial simplex fails the convex hull algorithm is recalculated by precise arithmetics provided by the iRRAM library [38].

4. Dynamic vertices

To change an existing triangulation according to a given application all intended operations, i.e. which vertices are deleted, are inserted, changed their vertex weight, or are displaced by a given vector are collected. Each process has a local list of changes that have to be performed on its own vertices. The changes are then applied successively in the order they are presented below. The sequential execution of the changes to the triangulation is required as the algorithms used to perform each change are not compatible for concurrent execution.

The updates of the triangulation cause also changes to the boundary simplices. The connections are automatically set as described in Section 2.3. Also new "internal surfaces" between processes can occur changing the topology of process communication. Any of the presented algorithms may create simplices which are shared between processes p_i and p_j that formerly did not share any simplex. This information always stems from a process p_k which is connected to the processes p_i and p_j. Following the message of process p_k, both unconnected processes exchange their vertices to establish a connection for communication (Section 2.3). Similarly overhead connections are removed after all changes have been completed by checking if communicating processes share any simplex at all. This is done to reduce both memory and communication over-

```
begin delete_vertices( dlist )
while dlist not empty do
  if messages received then
    call deletion_events()
  else
    vertex V ← V ∈ dlist
    for all simplices σ with V ∈ σ do
      if σ not blocked then
        block σ
      else
        V → dlist
        V ← V ∈ dlist
      end if
    end for
    if σ: V ∈ σ is shared then
      if rank > neighbor ranks then
        block σ
      else
        request neighbor to block σ
      end if
    else
      delete vertex and fill cavity
    end if
  end if
end while
end delete_vertices
```

```
begin deletion_events()
if blocking requested then
  if no access then
    deny blocking
  else
    block simplex
  end if
else if unblock requested then
  unblock simplex
else if blocking successful then
  if all simplices accessible then
    delete vertex and fill cavity
    send result to neighbors
  end if
else if access denied then
  unblock simplices
  send unblock request to neighbor
  V → dlist
  V ← V ∈ dlist
else if result received then
  if improper deletion history then
    receive delayed messages
  end if
  apply received changes
end if
end deletion_events
```

Fig. 7. Pseudo-code for deleting vertices from a list `dlist`. The left panel shows the serial part of the code. The right panel the response of the process to messages.

head. We do not further note this routine in the detailed description of the algorithms.

4.1. Vertex deletion

The vertex deletion algorithm is similar to the algorithm used for the initial construction. The deletion of a vertex V infers that all simplices having it as endpoint will be removed. The Delaunay triangles marking the border of the created cavity will then be stored in an AFL. The neighbor vertices of V defined by the edges of the original triangulation are stored in a vertex list VL. The AFL and the reduced vertex set VL is used in the same manner as the expansion step during the construction of the initial triangulation (Section 3). For simplicity we do not use a uniform grid for the vertex search here even though the algorithm scales like $\mathcal{O}(k^2)$ with k being the number of edges of the vertex V. In general k is small ($\lesssim 20$) and the loss of performance due to non-optimal scaling is negligible compared to the overhead caused by creation of the uniform grid.

The parallel scheme is a combination of task and data-parallelism. A process p_i trying to delete a vertex collects the incident simplices. If none of the simplices is a boundary simplex, the vertex is deleted like in the serial case (Fig. 7, left panel). Otherwise the simplices are marked as blocked and the corresponding neighbor processes p_k are contacted. These will check if:

(1) The simplices exist.
(2) The simplices are not blocked by other processes.

If both conditions are fulfilled the processes p_k block the simplices and send a success message to p_i. As soon as p_i has received success messages from all p_k it fills the cavity using the above mentioned incremental construction algorithm. The result is sent

to the processes p_k which performs the corresponding changes (Fig. 7, right panel). If the conditions are not fulfilled the process p_i will unblock the simplices. Consequently, further neighbor processes which successfully tested the conditions are requested to unblock the simplices, too. The vertex is then deferred and p_i tries to delete another vertex from the list of vertices to be removed as requested by the application. The vertex is reconsidered later when both simplex access conditions are fulfilled (Fig. 7, right panel).

The first condition can be violated by message delays (Section 2.3): Two cavities filled by the processes p_i and p_j may depend on each other, e.g., the action of p_j may occur after the changes performed by p_i. When now p_j requests the blocking of its cavity at a third process p_k the latter may not have applied the changes of p_i thus some simplices may not exist at p_k yet. The second condition can be violated if a neighbor process p_j deletes a vertex whose cavity intersects with the cavity of the vertex to be deleted by process p_i at the same time.

A significant amount of messages can be saved by modifying the two conditions above. Introducing a priority order between the processes based on their rank we test the second condition only locally when the processes sharing the simplex are of lower rank. Thus, any process assumes that it can access a simplex $\sigma_{\langle A,B,C,D\rangle}$ when its rank is the highest among the rank of the processes the vertices A, B, C, D are assigned to and if $\sigma_{\langle A,B,C,D\rangle}$ is not blocked (Fig. 7, left panel).

However, the above improvement can lead to a wrong message order due to message delays (Section 2.3). Therefore a receiving process p_i has to restore the correct message order: We incrementally send a history of vertices deleted by the sending process and by its neighbors. The receiving process p_i then searches for messages concerning not yet deleted vertices. This introduces some additional effort, however, the detection of the rather rare events of wrong message order is still much faster than the full test for both conditions.

Task-parallelism is achieved by using the idle times of process p_i while waiting for the response to the messages explained above. Priority is given to execute checks of simplices requested by neighbor processes. If none of those messages have to be dealt with, then process p_i tries to delete another vertex from the list.

The whole algorithm is finished as soon as a master process has recorded that all processes have deleted their vertex list.

4.2. Vertex insertion

The incremental vertex insertion requires two steps: Firstly, the determination of the topological location of the point within the triangulation and secondly computing the resulting triangulation. For the first task we use a randomized simplex visibility walk as described previously [18,34]. Since the simplex walk does not change the triangulation, it is parallelized directly: When a process encounters a simplex that locates the vertex beyond one of its boundary faces, it sends the vertex to one of the processes owning the vertices of that face. It will also send the simplex and one of its vertices to the receiving process. The receiver will start the simplex walk with that simplex or—if the simplex does not exist anymore (see below)—a simplex which is close to the given vertex. The vertex is assigned to the process that finishes the simplex walk thereby maintaining a good partition. The starting simplex is chosen to be the first simplex in the simplex list as long as the application provides no information of a close vertex (which is the case for modeling dividing cells). The expected complexity for locating the vertex in a uniform distribution is $\mathcal{O}(N^{1/3})$ (N is the number of vertices) as in the serial case [18,34].

For the insertion of the vertex in the triangulation we use the Bowyer–Watson algorithm [14,15]. All simplices which violate the Delaunay criterion when the vertex V is inserted are deleted and the cavity is re-triangulated using a star-shaped triangulation with center V (Fig. 8). For parallelization we use a concur-

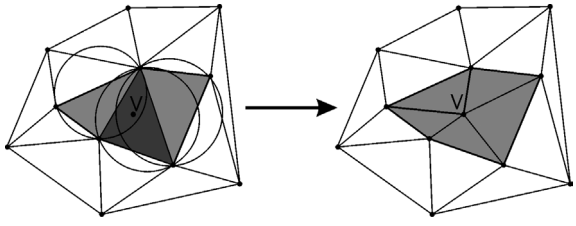

Fig. 8. Bowyer–Watson algorithm for an unweighted Delaunay triangulation. The simplex containing the vertex V (dark grey triangle) is found by the simplex walk. Then all simplices which do not fulfill the empty orthocircle criterion (left panel, grey triangles and circles) are replaced by a star shaped triangulation centered at V (right panel, grey triangles).

```
begin insert_vertices( ilist )
while ilist not empty do
  if messages received then
    call insert_events()
  else
    vertex V ← V ∈ ilist
    if simplex walk at border then
      send simplex walk to neighbor
    else
      start virtual domain vd
      block simplices invalidated by V
      construct virtual simplices
      if σ ∈ vd is shared then
        send vd to neighbor processes
      else
        turn virtual domain 'real'
      end if
    end if
  end if
end while
end insert_vertices
```

```
begin insert_events()
if simplex walk request then
  continue simplex walk
  if simplex walk at border then
    send simplex walk to neighbor
  else
    start virtual domain
  end if
else if virtual domain vd then
  if data structure not consistent then
    receive delayed messages
  end if
  expand virtual domain
  if vd is complete then
    send result to initiator of vd
  else if σ ∈ vd is blocked then
    send abort to initiator of vd
  end if
else if virtual domain complete then
  turn virtual domain 'real'
  request neighbor to turn vd 'real'
else if request to turn vd real then
  turn virtual domain 'real'
else if virtual domain blocked then
  abort virtual domain
  request neighbors to unblock vd
end if
end insert_events
```

Fig. 9. Pseudo-code for the insertion routine. The left panel shows the local process operations. The right panel shows the processes response to received messages.

rent Bowyer–Watson algorithm: vertex insertions are performed at different locations of the triangulation. For this purpose, *virtual domains* are introduced as the set of *virtual simplices* filling the star-shaped region that results from the Bowyer–Watson algorithm. Thus the virtual domain is the result of the insertion of the vertex V without actually applying the changes, i.e. deletion of invalidated simplices or upset of simplex-simplex-neighborship relations.

During the parallel routine the simplices invalidated by the vertex V are collected. These simplices are blocked (Fig. 9, left panel). A virtual simplex is constructed only if a neighbor σ_1 of an invalidated simplex σ_2 remains valid. Then the virtual simplex is constructed out of the three common endpoints of σ_1 and σ_2 and the vertex V. If some of the invalidated simplices are boundary simplices, the neighbor processes are contacted. When the simplices are not used in another insertion process, i.e. are blocked, then these processes perform three tasks (Fig. 9, right panel):

(1) Find further simplices invalidated by V.
(2) Construct virtual simplices if they do not exist yet.
(3) Contact further processes to do the same if required.

After the whole invalidated region has been determined the virtual domain is turned 'real' by erasing all invalid simplices and setting the correct neighborship relations thus completing the Bowyer–Watson algorithm on all involved neighboring processes. In order to do this, the initiating process will send a final 'completion' signal to all participating processes (Fig. 9), thus acting like a master process controlling its dynamically selected workers. In the case that simplices

are blocked by a concurrent virtual domain an abort signal is sent to the owner process of vertex V which abandons the virtual domain and orders other neighbor processes to do so as well. Note, that this might result in deadlocks, when two or more processes block each other repeatedly if all remaining vertices have intersecting virtual domains. To solve such deadlocks the process ranks are compared giving the higher-ranking process the exclusive access rights for the involved simplices until its vertex is inserted.

A careful analysis shows that two virtual domains which do not overlap but share at least one face can result in a non-regular triangulation. For example, let the simplices σ_1 and σ_2 invalidated by the vertices V_1 and V_2, respectively, share a common face. The assumption of non-overlapping virtual domains implies that neither vertex V_2 invalidates σ_1 nor V_1 invalidates σ_2. However, vertex V_2 may invalidate the new simplex formed by the common face of the two simplices σ_1 and σ_2 and the vertex V_1. In that way the two virtual domains can create non-regular simplices. Thus not only invalidated simplices but also their neighbors have to be blocked.

Similar to the deletion routine we achieve task-parallelism by using the idle times of a process to answer messages of neighbor processes or trying to insert other vertices. The advantage of using virtual domains is that the collection of invalidated simplices and the construction of new ones are done in parallel. Thus, upon successful completion no post-processing of the collected simplices is required.

A disadvantage of the used algorithm is the occurrence of message delays. In contrast to the deletion algorithm, the procedure to create the virtual domain provides already all necessary information to detect which vertices have to be inserted as the vertices invalidate different sets of simplices. If these simplices do not exist or are still virtual, the receiving process checks for the completion message of the initiating process (Fig. 9, right panel).

Again the completion of the whole parallel insertion algorithm is controlled by a master process.

The algorithm described here is quite similar to a task-parallel Bowyer–Watson algorithm proposed previously [19,20]. However, in the present work virtual domains have been introduced to decrease the number of tasks that require communication. Other changes are the extension to a different data structure that supports the needed algorithms, and the extension to regular triangulations.

5. Kinetic vertices

5.1. Artificial vertices in the flip algorithm

As we use the Lawson flip algorithm [13,16,17] to maintain the triangulation of kinetic vertices it is convenient to use artificial vertices (with zero weight) which ensure convexity of the triangulation during vertex movement. These artificial vertices are located far outside the triangulation so that they do not interfere with the application. The bucketing technique [32] used during the incremental construction has to be adapted as the grid resulting from the inclusion of the artificial vertices would be much too large to allow efficient scanning. Therefore the additional artificial vertices are considered only when the signed Delaunay distance of a given face minimized with grid vertices is larger than the minimum distance between the uniform grid and the artificial vertices or when no grid vertex can be found to expand that face.

5.2. Maintain valid triangulation

First of all, the algorithm ensures that the movement of a vertex does not change the orientation of a simplex, i.e. the position $\mathbf{v}$ of vertex V remains in the feasible region Ω_V defined by

$$\Omega_V = \big\{\mathbf{p} \in \mathbb{R}^3 \colon \mathrm{O}\big(A, B, C, P = (\mathbf{p}, 0)\big) > 0 \quad \forall \sigma_{\langle A,B,C,V\rangle} \in \mathcal{T}\big\}. \tag{11}$$

In order to do this, for every vertex a directional maximum displacement is calculated [18]. If the desired position change of the vertex is larger the movement is split into several steps. As the movement of one vertex influences how far the others are allowed to move, an asynchronous movement is necessary. To keep the maximum parallel performance, the parallel algorithm will only move vertices which have no connection to vertices owned by a process with a higher rank. The remaining vertices will only be moved after receipt of a message from the neighboring process about a performed move of the connected vertices. When every vertex has been moved the flip algorithm is applied. If a vertex has been displaced only in parts then

```
begin move_vertices( mlist, wlist )
perform weight changes from wlist
while mlist not empty do
  for all V ∈ mlist do
    if orientation violated then
      split displacement
    else
      displace vertex
      remove vertex from mlist
    end if
  end for
  call do_flips()
end while
end move_vertices
```

Fig. 10. The pseudo-code for the kinetic algorithm moving vertices according to `mlist` and performs weight changes according to `wlist`.

the orientation check is repeated, another part of the movement is performed and the flip algorithm is called again. This procedure is repeated until all vertices have moved to their final position (Fig. 10).

Instead of calculating the maximum displacements the program can be configured such that an *a posteriori* check is applied to detect misoriented simplices. If such simplices are found total re-triangulation becomes necessary. Nevertheless the average calculation time needed may be reduced by replacing the movement splitting calculation by the faster *a posteriori* orientation control. The choice of the appropriate method depends on the length of the spatial stepsize of the application.

5.3. Parallel flip algorithm

The sequential algorithm is a simple search for non-regular simplices in a list of simplices: A simplex of that list is chosen which we call an *active* simplex. The Delaunay criterion with its neighbor simplices (the *passive* simplices) is calculated. If a flip is necessary it will be performed, and the new simplices are stored at the end of the simplex list [3,10,13,16]. The algorithm iterates through the list. When the end is reached the triangulation has become regular [10]. As every valid simplex will become an active simplex at least once it is possible that the same Delaunay criterion is computed twice to check for the flip $2 \to 3$, three times for flip $3 \to 2$, and four times for flip $4 \to 1$. To prevent this overhead every result of a Delaunay criterion with a neighbor simplex is stored at both the active and passive simplices.

Our parallel algorithm is based on the definition of access rights to the simplices. The basic idea is to use a relational operation of vertices. A process p has access to a simplex $\sigma_{\langle A,B,C,D\rangle}$ when p has the greatest numbers of vertices in $\sigma_{\langle A,B,C,D\rangle}$ (4, 3, or 2). If several processes have equal greatest numbers of vertices in $\sigma_{\langle A,B,C,D\rangle}$ (all 1 or all 2) the one with the 'largest' vertex (using an arbitrary 'less than' relation) gains access rights. The definition does not require any communication, i.e. any process can determine in advance which simplex is handled by which process. Consequently, blocking of simplices is now a purely local operation. Additionally, this definition of access rights has the favorable implication, that every flip affects only simplices accessible by exactly one or two processes even when four simplices are involved.

In the algorithm a process checks every simplex with or without having access rights. Any flip of accessible simplices will be performed by the process. If boundary simplices are involved the result will be stored to inform neighbor processes and will be exchanged regularly between the processes (Fig. 11, left panel). Every process uses the received information to apply the flip sequence generated by the corresponding sending process to its copy of the boundary simplices until the final state is achieved. No further communication is required.

If a flip between accessible and non-accessible simplices is detected by a process p_1 the flip will be *suggested* to the other process p_2 (Fig. 11). That process will check if this flip is possible which is the case if the simplex has not already been involved in another flip and not been suggested by p_2 itself. All possible flips are performed immediately. To prevent deadlocks of processes suggesting each other the same flip, every flip is made unique by process ranking.

A special case is that a flip may not be detected by the processes p_1 and p_2 owning the simplices. They may not share all the simplices or they just do not have the corresponding simplex–simplex neighborship relations (Fig. 12). The detecting process then suggests a flip to one of these processes p_1 and p_2. The receiving process will detect if this flip is still possible and suggest the flip to the second process like in the case of normally suggested flips (Fig. 11, right panel).

```
begin do_flips()
slist {list of simplices to check}
flist {list of flips}
while slist not empty do
  for i = 0 until i < frequency do
    if σ ∈ slist is non-regular then
      if all simplices accessible then
        execute flip
        attach new simplices to slist
        if flip is shared then
          store result in flist
        end if
      else
        block simplices
        store suggested flip in flist
      end if
    end if
  end for
  send flist to neighbors
  call flip_events()
end while
end do_flips
```

```
begin flip_events()
count message from neighbor process
if flip counting not up to date then
  received delayed messages
end if
if flip result then
  apply changes
else if flip suggested then
  if simplices accessible then
    apply flip
    store result in flist
  else
    reject flip
  end if
else if flip suggested by third then
  if simplices accessible then
    store suggested flip in flist
  end if
else if reject suggested flip then
  unblock simplices
end if
end flip_events
```

Fig. 11. The pseudo-code for the parallel flip algorithm. The left panel shows the serial part of the algorithm while the right panel denotes the events following received messages. The term 'frequency' determines the number of simplices which are checked for the Delaunay criterion before parallel tasks are performed. The term 'suggested by third' refers to Fig. 12.

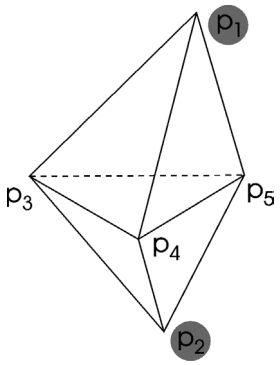

Fig. 12. An example of a suggested flip that cannot be detected by the processes with access rights to either of the simplices. The flip $2 \rightarrow 3$ shall be detected. The upper simplex belongs to process p_1 and the lower one to p_2 (indicated by the grey shading). Both processes are not linked because they do not share any simplex (compare Fig. 4). Thus one of the processes p_3, p_4, or p_5 will detect and suggest the flip either to p_1 or p_2. The implementation assures that the flip is uniquely suggested by only one process.

Controlling the termination of the algorithm is widely similar to vertex deletion and insertion. In addition the master will request from every process a confirmation of the termination status. This final confirmation is required because a process p_i can report that it is in the finished state while a neighbor process p_k induces new non-regular simplices for the process p_i.

The loose synchronization of the processes raises the same difficulties message ordering as in the dynamic algorithms. Similarly as for vertex deletion this is detected by the receiving process. Each group of flips exchanged with neighbor processes will be counted at the sending and receiving end. Additionally, the sending process p_1 will send the counter of all flip messages it has received from its neighbor processes p_i. The receiver p_2 will then check if all flip messages from its neighbors p_k have been received according to the counters. If not, the message will be received and the correct message order is restored. Note that this message synchronization takes place only when the messages depend on each other, i.e. only processes which are neighbors of both processes p_1 and p_2 need to be considered for delayed messages. Otherwise the basic principle 'first come, first served' is applied.

5.4. Vertex movements

To increase the overall performance, we replace the movement operation by a combination of vertex deletion and insertion if the movement has to be split in too many parts. As this might happen to several vertices, these events are collected and performed at once when all other vertices have already been moved with the flip routine.

During the iterations of the flip algorithm non-flippable configurations can be generated [10,11]. Such configurations are composed of non-regular simplices where no subset is in a simplex configuration required for the elementary topological transformations: The simplices are in a non-convex configuration (for an example see [11]). Thus the flip algorithm completes without restoring the Delaunay property of the triangulation. Experimental results show that this restricts the general applicability of the algorithm (Section 6.5).

5.5. Changing vertex weights

In our target application the agent-based simulation of biological tissues cells may change their size according to some law. The size of the cell is associated with the weight of a vertex. For realistic models small deviations $\delta w_v \ll w_v$ of the vertex weights w_v are assumed. These changes generally have only minor effects on the triangulation and are therefore applied before the kinetic routine starts (Fig. 11, left panel). The topological changes induced by the new vertex weights are applied together with the changes resulting from the vertex displacements when the flip algorithm is performed during the kinetic routine. The performance of this routine is included in the movement of vertices and will not be analyzed separately.

6. Experimental results

We tested each of the algorithms separately using the following protocols. The test systems were uniform vertex distributions inside a cube. Distributions with 1000 (1k), 4000 (4k), 8000 (8k), 16000 (16k), 100000 (100k), and 1000000 (1000k) vertices were used. Most of the tests were performed with usual Delaunay triangulations without vertex weights. The weight has major influence on the initial construction and is tested for the system size of 16k. The performance of the dynamic and kinetic algorithms is widely independent of the presence of vertex weights—provided the weight changes are small. The regular triangulation is tested with a uniform distribution of the vertex weights with values between 0, and 10, 100, or 1000% of the average vertex distance. The average vertex distance l is approximated by $l = (V_{\text{cube}}/N_{\text{vert}})^{1/3}$ with V_{cube} the volume of the cube and N_{vert} the numbers of generated vertices. Note, that redundant vertices may be included in N_{vert}. Any operation on the triangulation can change the balance of vertices distributed across the processes. In the tests shown the indicated change started with a fresh partition. Currently dynamic load balancing between the processes is left to the application.

The initial triangulation and the changes in terms of vertex positions, insertions, deletions, and displacements were generated using the pseudo-random number generator ‘Scalable Parallel Random Number Generator’ (SPRNG) version 1.0 [39].

The tests were performed on an Origin 3800 with R12000 processors located at the Centre for Information Technology Services and High Performance Computing at the Dresden University of Technology. Times were recorded using `MPI::Wtime()` command from the MPI library. Error bars indicate one standard deviation out of 10 different experiments with different seeds for SPRNG. Data points missing in the figures have not been measured due to either large execution times on small numbers of processors, or memory constraints for large sizes, or too few vertices to be distributed efficiently on many processors.

All presented algorithms show a remarkable scaling behavior, i.e. the execution time is decreasing when more processors are used. An overview of the parallel performance is shown in Table 1 as a fit to Amdahl’s law [40].

6.1. Initial construction

Although the modified incremental construction algorithm exhibits a sufficient scaling behavior (Fig. 13), the cost for inserting a single vertex is high. The execution time of the algorithm increases dramatically when the vertex weights are large compared to the average

Table 1
Table of parallel efficiency for the various operations performed on the triangulation. The experiments were fitted with Amdahl's law [40]: $t_p = t_s[(1-f)+f/p]$, with t_p the parallel execution time, t_s the execution time on a single processor, f the fraction of parallelisable work, and p the number of processors used. Shown are f (t_s) with t_s in seconds. Optimal scaling behavior would be $f = 1$. Data points at 1000k vertices and 1000% insertion are missing for hardware constraints

Operation	1k	4k	8k	16k	100k	1000k
Deletion						
1%	0.70 (0.02)	0.80 (0.09)	0.86 (0.16)	0.94 (0.37)	0.98 (2.5)	1.00 (26.6)
10%	0.88 (0.19)	0.93 (0.82)	0.97 (1.8)	0.98 (3.7)	0.99 (25.5)	1.00 (258)
100%	0.94 (1.7)	0.97 (8.1)	0.99 (17.6)	0.99 (36.2)	0.99 (246)	1.00 (2660)
Insertion						
10%	0.37 (0.05)	0.72 (0.26)	0.79 (0.58)	0.75 (1.1)	0.85 (5.6)	0.97 (109)
100%	0.69 (0.63)	0.86 (3.1)	0.90 (6.6)	0.93 (14.3)	0.96 (89.3)	0.99 (1274)
1000%	0.79 (7.7)	0.90 (34.7)	0.95 (77.9)	0.95 (155)	0.97 (956)	
Random insertion						
10%	0.56 (0.06)	0.74 (0.27)	0.86 (0.68)	0.91 (1.6)	0.97 (14.7)	0.99 (198)
100%	0.63 (0.74)	0.83 (3.2)	0.90 (7.6)	0.93 (16.6)	0.98 (156)	0.99 (2377)
1000%	0.75 (8.4)	0.87 (40.1)	0.94 (91)	0.95 (197)	0.98 (1790)	
Displacement						
10%	0.65 (0.37)	0.82 (1.6)	0.85 (3.3)	0.87 (6.7)	0.83 (24.0)	0.96 (442)
100%	0.64 (1.8)	0.84 (8.4)	0.87 (17.6)	0.89 (36.4)	0.90 (175)	
1000%	0.37 (10.0)	0.61 (54.6)	0.77 (149)	0.79 (313)	0.83 (1975)	

vertex distance (Fig. 13). The time loss is caused by the increased numbers of vertices to be checked during face expansion.

6.2. Insertion

To test the dynamical insertion of vertices initial triangulations were constructed and vertices were added in two manners: First, the new vertices were inserted close (less than the average vertex distance) to other vertices—mimicking cell division in biological tissues. This is used to provide an initial guess where to start the simplex visibility walk. Second, vertices were added randomly inside the cube containing the initial vertex distribution without a guess. Three different numbers of vertices were chosen: (i) A fraction 0.1 of the initial system size (10% insertion). (ii) The vertex number was doubled (100% insertion). (iii) The 10-fold numbers of vertices of the initial triangulation was added (1000% insertion).

The parallel scaling factor of the insertion algorithm shows no significant difference between localized and random vertex insertion (Fig. 14, Table 1). For small system size the parallel scaling is reduced, since much of the triangulation consists of boundary simplices requiring much communication. In the case of weighted vertices the insertion times are decreasing with increasing weight (not shown). This effect is caused by an increased amount of redundant vertices. For those the algorithm stops with the completion of

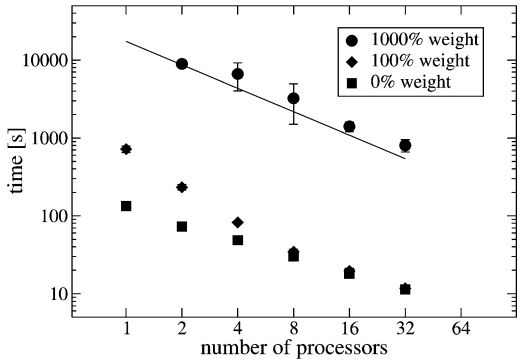

Fig. 13. Parallel performance of the incremental construction algorithm for 16k vertices with different closed symbols denoting different vertex weights. The line indicates the slope of −1 achieved by optimal scaling for comparison. (The data for 1000% vertex weight have not been measured with 1 processor and contain less than 10 runs per data point for 2, 4, and 8 processors due to extremely long execution times.)

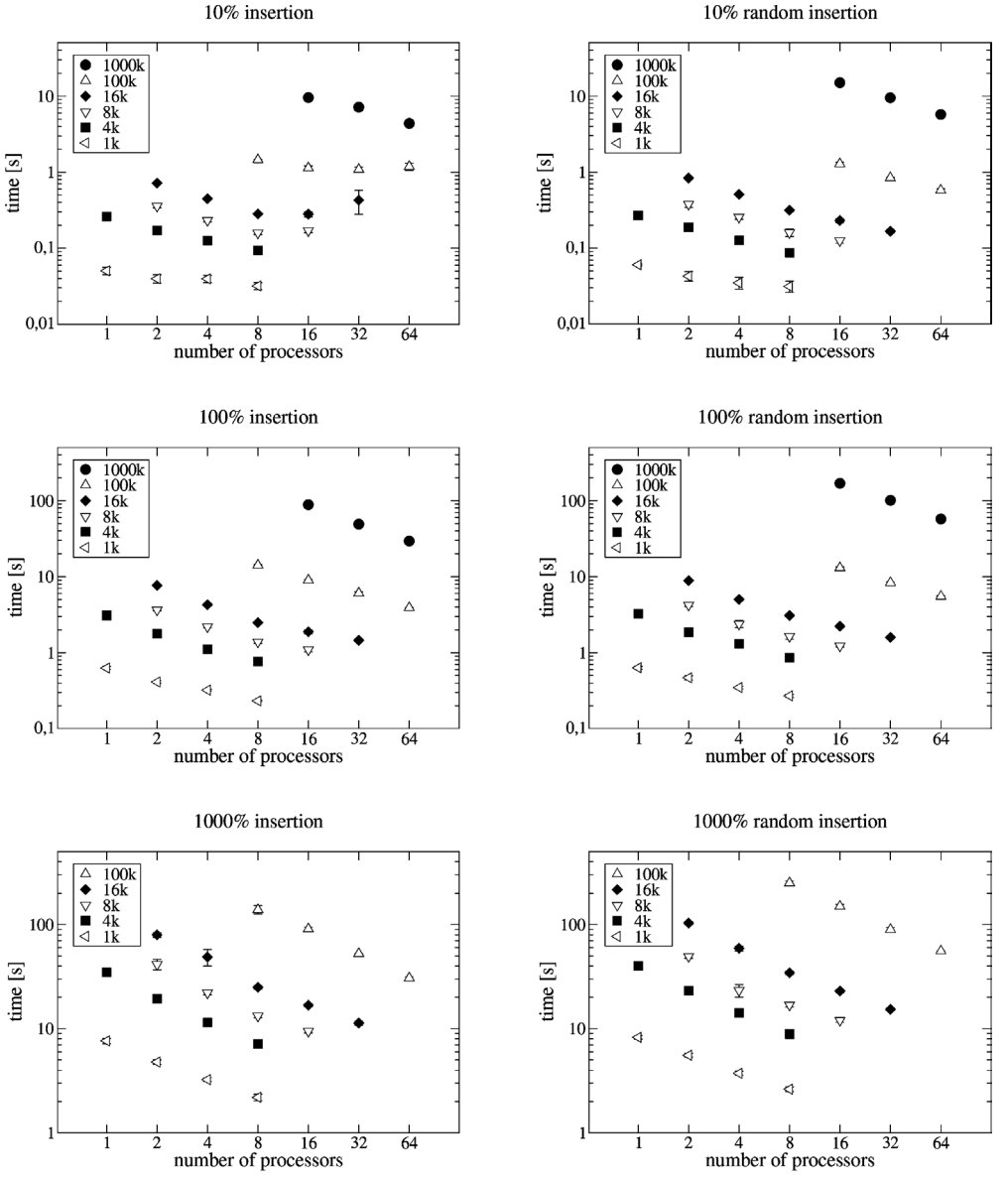

Fig. 14. Timings of the insertion routine for inserting vertices close to a given vertex (left panels) and without providing an initial guess (right panels). The different symbols denote different initial system sizes while the different panels correspond to different fractions of inserted vertices. See text for details.

the simplex walk omitting the effort of the Bowyer–Watson algorithm.

Despite similar scaling behavior the timing when inserting vertices without initial guess is significantly increased for large sizes indicating the additional time required to locate the first simplex of the Bowyer–Watson algorithm by the simplex visibility walk.

6.3. Combining initial construction and insertion

As the initial construction has an unsatisfying serial performance we performed tests in which 1000 vertices per processor were selected randomly for triangulation using the incremental construction while the remaining ones (if any) are inserted using the concur-

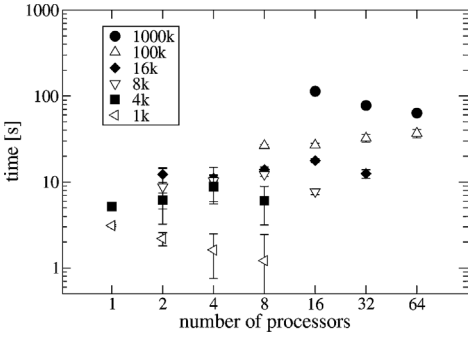

Fig. 15. The construction times for the initial triangulation using incremental construction and concurrent Bowyer–Watson combined. The different symbols denote different system sizes. Error bars indicate one standard deviation of 100 runs. The construction times are substantially smaller as compared to a pure incremental construction (Fig. 13). However, the scaling behavior is destroyed. See text for details.

rent Bowyer–Watson algorithm. The number of vertices has to be chosen sufficiently large in order to distribute them on large numbers of processes.

The overall gain in performance by the combined initial construction is significant as long as a relevant amount of vertices are inserted with the concurrent Bowyer–Watson algorithm (see Fig. 15 as compared to Fig. 13). However, the scaling behavior is destroyed. The more processors are used the more vertices are triangulated by the incremental construction algorithm and less by the concurrent Bowyer–Watson algorithm. Therefore the gain achieved by the parallel execution is counteracted by a higher fraction of vertices inserted with the slower incremental construction. The total gain in performance can also be observed when triangulating vertices with different weights. For small weights there is no significant difference to the unweighted case. For large weights the overall construction time is dominated by the slow incremental construction step at the beginning (not shown).

6.4. Deletion

The deletion algorithm was tested by removing a randomly chosen fraction of 1, 10, and 100% of the triangulated vertices. It exhibits the best scaling behavior of the algorithms presented here (Table 1). Smaller gains in performance are observed for small systems and small numbers of deleted vertices (Fig. 16). This

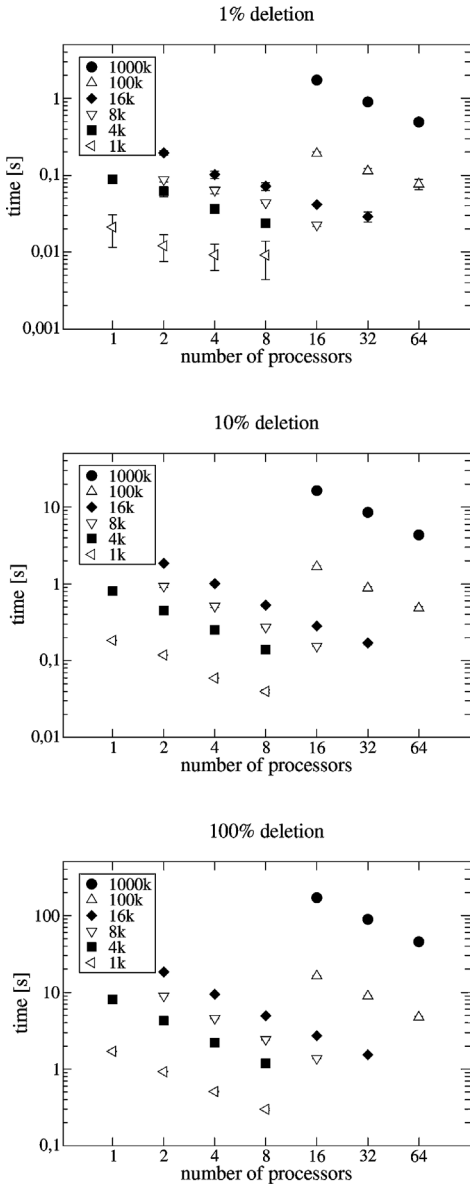

Fig. 16. Performance graph of the deletion routine. Different symbols denote different initial system size. Different panels correspond to different fractions of deleted vertices. See text for details.

result reflects the greater influence of inhomogeneities of the data distribution within data-parallel algorithms. The time needed to delete all the vertices is about as large as the construction time for the triangulation (Fig. 15). The cause is the slow incremental construction algorithm used to fill the cavity which exceeds the time needed to locate the vertex in the tri-

angulation with the concurrent Bowyer–Watson algorithm.

6.5. Moving vertices

In the kinetic tests all vertices have been moved by choosing every component of the displacement vector uniform randomly within a symmetric interval $[-l, l]$ with l equal 10, 100, or 1000% of the average distance of vertices (Fig. 17). For most applications the smallest displacement is realistic. The numerical experiments also support the view that larger changes in the triangulation are inefficiently solved by the flip algorithm. As mentioned before the problem of non-flippable configurations can arise when moving all vertices synchronously. Indeed, for 100% displacement this occurred quite frequently while in the case of 1000% displacement almost none of the new configurations could be achieved by the flip algorithm and the run got stuck in non-flippable configurations. When the code detects such situations, the triangulation is recomputed form scratch. Therefore the presented timings consist of the time needed for the flip algorithm until failure plus the time for a complete re-triangulation (Fig. 17). Thus, only displacements significantly smaller than the average vertex distance take advantage of the flip algorithm to restore the Delaunay criterion. Note that, all times shown include the time for checking the orientation of simplices before applying the vertex displacements (Section 5.2) and the *a posteriori* orientation check.

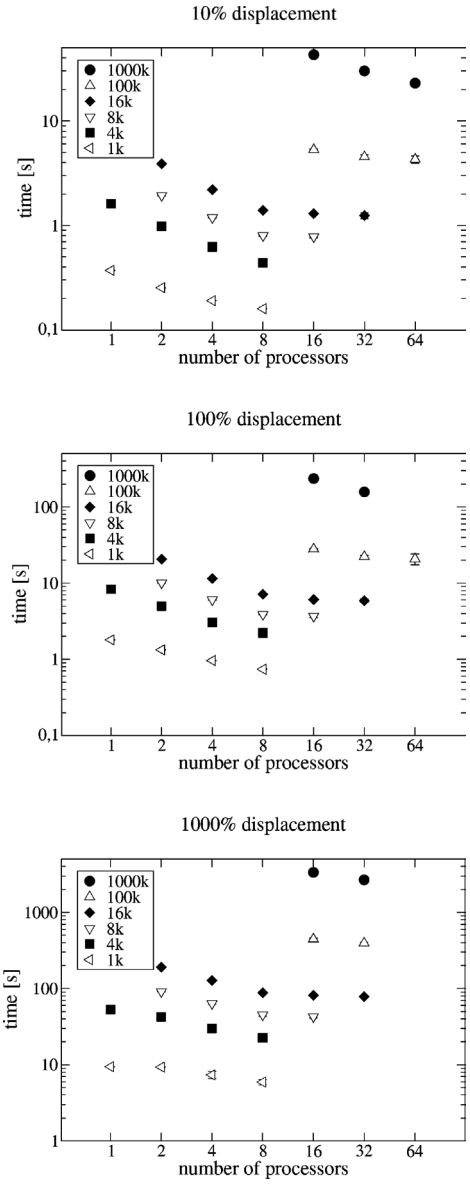

Fig. 17. Computing times of the kinetic routine. The different panels correspond to variations of the displacement length while the symbols denote different sizes of the triangulation. The kinetics algorithm for its own is represented by the graph for 10% displacement. Other panels intermix other parts of the algorithms. See text for details.

7. Conclusions

We presented a tool that offers the parallel execution of particle dynamics on parallel machines on top of a regular triangulation. To our knowledge, the parallel algorithms for dynamic and kinetic vertices are completely new and for the first time permit the full set of all necessary topological operations on regular triangulations performed on a distributed memory architecture.

The assumption of uniform vertex distribution affects only the initial triangulation, whereas the dynamic and kinetic algorithms do not rely on this assumption. Inhomogeneities in the distribution of work load concerning the number of insertions, deletions, displacements, or weight changes should be counterbalanced by a proper partition of data provided by the user. The dynamic and kinetic algorithms presented here are able to deal with any data distribution al-

though the performance suffers in some particular configurations.

For many vertex distributions it important to provide numerical robustness in all Delaunay algorithms. The robustness can be dropped by the user in favor of faster but insecure floating point operations if the vertex distributions and changes applied to the triangulation exhibit sufficient small computational errors of the signed Delaunay distance (Eq. (5)), orientation (Eq. (3)) and orthosphere (Eq. (4)) primitive.

A drawback of the tool presented here is the relative expensive data structure that has to be maintained to serve the needs of many applications (a typical example of 100k vertices on 16 CPUs requires about 48 MB per CPU). In spite of the cost of the initial triangulation paid only once, the dynamic and kinetic routines show a reasonable performance increase upon parallel execution. Especially, when the tool is applied to biological tissues the optimal requirements of small displacements can be assumed leading to a substantial performance gain for the desired operations on the Delaunay graph.

During all vertex operations (except deletion) redundant vertices might be generated which are handled properly by the triangulation. It is up to the application to treat these vertices. For biological simulations however, redundant vertices are unsuitable as they do not represent any physiological behavior. Thus, the application should prevent the generation of redundant vertices by setting a proper position or weight. A sufficient condition is that any sphere associated to a vertex is never covered by more than one half by another sphere [3].

As this work is focused on the dynamic and kinetic changes to an existing regular triangulation the performance of the initial construction is of minor importance. The major improvements are for the dynamical changes of an existing triangulation. In the context of agent-based cell tissue modeling three scenarios are of importance: First, the case of fast dividing cells. Here, the performance of the insertion algorithm is important in order to insert new cells. Second, the death of cells is important and depends on the deletion routine removing dead cells. The third scenario are fast moving cells relying on the kinetic routine. For the simulation of cells the two dynamic and the kinetic routines will be executed many times thus requiring most of the calculation time. The application of the presented algorithms to single-cell-based models of biological tissue is a matter of further investigations.

Appendix A. Computation of the signed Delaunay distance

The signed Delaunay distance $\mathrm{SD}(D)$ of a vertex $D = (\mathbf{d}, w_d)$ in respect to the face $\langle A, B, C\rangle$ is defined as [21,22]:

$$\mathrm{SD}(D) \stackrel{\text{def}}{=} \left[\mathbf{m}(D) - \mathbf{m}_{\langle A,B,C\rangle}\right] \cdot \frac{(\mathbf{a}-\mathbf{b})\times(\mathbf{a}-\mathbf{c})}{\|(\mathbf{a}-\mathbf{b})\times(\mathbf{a}-\mathbf{c})\|} \tag{A.1}$$

by replacing the circumsphere centers $\mathbf{m}(\mathbf{d})$ and $\mathbf{m}_{\langle \mathbf{a},\mathbf{b},\mathbf{c}\rangle}$ by the corresponding orthosphere centers $\mathbf{m}(D)$ and $\mathbf{m}_{\langle A,B,C\rangle}$ in Eq. (5). To get a useful expression for evaluating $\mathrm{SD}(D)$ we rewrite the orthosphere equation (Eq. (2)) in matrix form following [22]:

$$\begin{bmatrix} 1 & a_x & a_y & a_z \\ 1 & b_x & b_y & b_z \\ 1 & c_x & c_y & c_z \\ 1 & d_x & d_y & d_z \end{bmatrix} \begin{bmatrix} \|\mathbf{m}(D)\|^2 - w_m \\ -2m_x(D) \\ -2m_y(D) \\ -2m_z(D) \end{bmatrix} = -\begin{bmatrix} \|\mathbf{a}\|^2 - w_a \\ \|\mathbf{b}\|^2 - w_b \\ \|\mathbf{c}\|^2 - w_c \\ \|\mathbf{d}\|^2 - w_d \end{bmatrix}. \tag{A.2}$$

Subtracting the first row from the other rows gives

$$\begin{bmatrix} b_x - a_x & b_y - a_y & b_z - a_z \\ c_x - a_x & c_y - a_y & c_z - a_z \\ d_x - a_x & d_y - a_y & d_z - a_z \end{bmatrix} \begin{bmatrix} -2m_x(D) \\ -2m_y(D) \\ -2m_z(D) \end{bmatrix} = -\begin{bmatrix} \|\mathbf{b}\|^2 - \|\mathbf{a}\|^2 - w_b + w_a \\ \|\mathbf{c}\|^2 - \|\mathbf{a}\|^2 - w_c + w_a \\ \|\mathbf{d}\|^2 - \|\mathbf{a}\|^2 - w_d + w_a \end{bmatrix}. \tag{A.3}$$

We now write the equations in matrix form $M^{\mathrm{T}}[-2\mathbf{m}(D)] = -\boldsymbol{\alpha}$ with

$$M = \begin{bmatrix} b_x - a_x & c_x - a_x & d_x - a_x \\ b_y - a_y & c_y - a_y & d_y - a_y \\ b_z - a_z & c_z - a_z & d_z - a_z \end{bmatrix} \quad \text{and}$$

$$\boldsymbol{\alpha} = \begin{bmatrix} \|\mathbf{b}\|^2 - \|\mathbf{a}\|^2 - w_b + w_a \\ \|\mathbf{c}\|^2 - \|\mathbf{a}\|^2 - w_c + w_a \\ \|\mathbf{d}\|^2 - \|\mathbf{a}\|^2 - w_d + w_a \end{bmatrix}.$$

By finding a QR-decomposition [41] of the matrix M into an upper triangular matrix R and an orthogonal

matrix $Q = [\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3]$ with the orthonormal column vectors $\mathbf{q}_i$:

$$M = \begin{bmatrix} b_x - a_x & c_x - a_x & d_x - a_x \\ b_y - a_y & c_y - a_y & d_y - a_y \\ b_z - a_z & c_z - a_z & d_z - a_z \end{bmatrix} = QR$$

$$= \begin{bmatrix} q_{11} & q_{12} & q_{31} \\ q_{12} & q_{22} & q_{32} \\ q_{13} & q_{23} & q_{33} \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ 0 & r_{22} & r_{23} \\ 0 & 0 & r_{33} \end{bmatrix}. \quad \text{(A.4)}$$

We write the left-hand side of the equation as

$$M^{\mathrm{T}}\big[-2\mathbf{m}(D)\big] = R^{\mathrm{T}}\big[-2Q^{\mathrm{T}}\mathbf{m}(D)\big] = R^{\mathrm{T}}\boldsymbol{\mu} \quad \text{(A.5)}$$

with $\boldsymbol{\mu} = -2Q^{\mathrm{T}}\mathbf{m}(D)$. The vectors $\mathbf{q}_1$ and $\mathbf{q}_2$ can be chosen in the affine plane containing the face $\langle A, B, C\rangle$. Consequently, only $\mathbf{q}_3$ depends on the vertex D.

When comparing M with Eq. (3) one can see that $\det(M) = -\mathrm{O}(A, B, C, D)$, hence $\mathrm{O}(A, B, C, D) = -r_{11}r_{22}r_{33}$ when the QR decomposition is used. Therefore the sign of the determinant of the matrix R will give the orientation of the simplex formed by the face $\langle A, B, C\rangle$ and the vertex D.

Now we can rewrite the expression for the center of the orthosphere $\mathbf{m}(D)$ using Eq. (A.5) with only μ_z depending on the vertex D:

$$\mathbf{m}(D) = -\frac{1}{2}(\mu_x\mathbf{q}_1 + \mu_y\mathbf{q}_2 + \mu_z\mathbf{q}_3). \quad \text{(A.6)}$$

The center of the orthocircle of the face $\langle A, B, C\rangle$ is then given by

$$\mathbf{m}_{\langle A,B,C\rangle} = -\frac{1}{2}(\mu_x\mathbf{q}_1 + \mu_y\mathbf{q}_2) + (\mathbf{q}_3 \cdot \mathbf{a})\mathbf{q}_3. \quad \text{(A.7)}$$

Taking into account that:

$$\frac{(\mathbf{a}-\mathbf{b}) \times (\mathbf{a}-\mathbf{c})}{\|(\mathbf{a}-\mathbf{b}) \times (\mathbf{a}-\mathbf{c})\|} = \operatorname{sign}(r_{11}r_{22}r_{33})\mathbf{q}_3, \quad \text{(A.8)}$$

this yields another form of the signed Delaunay distance

$$\mathrm{SD}(D) = -\operatorname{sign}(r_{11}r_{22}r_{33})\left(\frac{\mu_z}{2} + \mathbf{q}_3 \cdot \mathbf{a}\right). \quad \text{(A.9)}$$

To find a vertex D, which expands the face $\langle A, B, C\rangle$, the QR decomposition for this face is performed once giving the three vectors $\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3$ and the components r_{11}, r_{12}, r_{22}. Now the signed Delaunay distance is evaluated by computing r_{33} to get the orientation primitive and μ_z for the orthosphere primitive.

References

[1] F. Aurenhammer, Voronoi diagram—a survey of a fundamental geometric data structure, ACM Comput. Surv. 23 (3) (1991) 345–405.

[2] A. Okabe, B. Boots, K. Sugihara, S.N. Chiu, Spatial Tessellations: Concepts and Applications of Voronoi Diagrams, second ed., Probability and Statistics, Wiley, New York, 2000.

[3] J.-A. Ferrez, Dynamic triangulations for efficient 3D simulation of granular materials, Ph.D. thesis, Ecole Polytechnique Federal de Lausanne, 2001.

[4] Y. Jiang, H. Levine, J. Glazier, Possible cooperation of differential adhesion and chemotaxis in mound formation of Dictyostelium, Biophys. J. 75 (6) (1998) 2615–2625.

[5] M. Meyer-Hermann, A mathematical model for the germinal center morphology and affinity maturation, J. Theor. Biol. 216 (3) (2002) 273–300.

[6] J.C. Dallon, H.G. Othmer, How cellular movement determines the collective force generated by the Dictyostelium discoideum slug, J. Theor. Biol. 231 (2) (2004) 203–222.

[7] F.A. Meineke, C.S. Potten, M. Loeffler, Cell migration and organization in the intestinal crypt using a lattice-free model, Cell Prolif. 34 (4) (2001) 253–266.

[8] E. Palsson, H.G. Othmer, A model for individual and collective cell movement in Dictyostelium discoideum, Proc. Natl. Acad. Sci. 97 (19) (2000) 10448–10453.

[9] G. Schaller, M. Meyer-Hermann, Multicellular tumor spheroids in an off-lattice Voronoi/Delaunay cell model, Phys. Rec. E 71 (2005) 051910.

[10] H. Edelsbrunner, N.R. Shah, Incremental topological flipping works for regular triangulations, Algorithmica 15 (3) (1996) 223–241.

[11] M. Vigo, N. Pla, Regular triangulations of dynamic sets of points, Comput. Aided Geom. Design 19 (2) (2002) 127–149.

[12] T. Beyer, M. Meyer-Hermann, G. Soff, A possible role of chemotaxis in germinal center formation, Inter. Immunol. 14 (12) (2002) 1369–1381.

[13] C.L. Lawson, Generation of a triangular grid with applications to contour plotting, Tech. Rep., Memo 299, Jet Propulsion Laboratory, Pasadena, CA, 1972.

[14] A. Bowyer, Computing Dirichlet tesselations, Comput. J. 24 (2) (1981) 162–166.

[15] D.F. Watson, Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes, Comput. J. 24 (2) (1981) 167–172.

[16] B. Joe, Three dimensional triangulations from local transformations, SIAM J. Sci. Stat. Comput. 10 (1989) 718–741.

[17] B. Joe, Construction of three-dimensional Delaunay triangulations using local transformations, Comput. Aided Geom. Design 8 (1991) 123–142.

[18] G. Schaller, M. Meyer-Hermann, Dynamic Delaunay tetrahedralizations and Voronoi tesselations in three dimensions, Comput. Phys. Comm. 162 (2004) 9–23.

[19] N. Chrisochoides, D. Nave, Parallel Delaunay mesh generation kernel, Internat. J. Numer. Methods Engrg. 58 (2) (2003) 167–176.

[20] N. Chrisochoides, F. Sukup, Task parallel implementation of the Bowyer–Watson algorithm, Tech. Rep. CTC96TR235, Cornell Theory Center, 1996.
[21] P. Cignoni, C. Montani, R. Pereo, R. Scopino, Parallel 3D Delaunay triangulation, Comput. Graphics Forum 12 (3) (1993) 129–142.
[22] Y.A. Teng, F. Sullivan, I. Beichl, E. Puppo, A data-parallel algorithm for three-dimensional Delaunay triangulation and its implementation, in: Proc. of Supercomputing, 1993, pp. 112–121.
[23] G.E. Blelloch, J.C. Hardwick, G.L. Miller, D. Talmor, Design and implementation of a practical parallel Delaunay algorithm, Algorithmica 24 (3) (1999) 243–269.
[24] S. Lee, C.-I. Park, C.-M. Park, An improved parallel algorithm for Delaunay triangulation on distributed memory parallel computers, Parallel Process. Lett. 11 (2–3) (2001) 341–352.
[25] I. Kolingerova, J. Kohout, Optimistic parallel Delaunay triangulation, Visual Comp. 18 (6) (2002) 511–529.
[26] F. Aurenhammer, Power diagram: properties, algorithms and applications, SIAM J. 16 (1) (1987) 78–96.
[27] K.Q. Brown, Voronoi diagrams form convex hulls, Inform. Process. Lett. 9 (5) (1979) 223–228.
[28] E.P. Mücke, A robust implementation for three-dimensional Delaunay triangulations, Internat. J. Comput. Geom. Appl. 8 (2) (1998) 255–276.
[29] Message Passing Interface Forum, MPI: A Message-Passing Interface Standard, Tech. Rep. CS-94-230, University of Tennessee, Knoxville, TN, 1994.
[30] D.H. McLain, Two dimensional interpolation from random data, Comput. J. 19 (2) (1976) 178–181.
[31] M. Tanemura, T. Ogawa, N. Ogita, A new algorithm for three-dimensional Voronoi tessellation, J. Comput. Phys. 51 (1983) 191–207.
[32] T. Asano, M. Edahiro, H. Imai, M. Iri, Practical use of bucketing techniques in computational geometry, in: G.T. Toussaint (Ed.), Computational Geometry, Elsevier, 1985.
[33] G. Karypis, V. Kumar, A coarse-grain parallel formulation of a multilevel k-way graph partitioning algorithm, in: Proc. 8th SIAM Conf. on Parallel Processing for Scientific Computing.
[34] O. Devillers, S. Pion, M. Teillaud, Walking in a triangulation, IJFCS 13 (2) (2002) 181–199.
[35] R.E. Moore, Interval Analysis, Prentice-Hall, Englewood Cliffs, NJ, 1966.
[36] Boost Library, URL http://www.boost.org.
[37] J.R. Shewchuk, Adaptive precision floating-point arithmetic and fast robust geometric predicates, Discrete Comput. Geom. 18 (2) (1997) 305–363.
[38] N.T. Müller, The iRRAM: Exact arithmetic in C++, in: Selected Papers from the 4th International Workshop on Computability and Complexity in Analysis, Springer-Verlag, 2001, pp. 222–252.
[39] Scalable Parallel Pseudo Random Number Generator Library, URL http://sprng.cs.fsu.edu.
[40] G.M. Amdahl, Validity of the single processor approach to achieving large scale computing capabilities, in: AFIPS Conf. Proc., 1967.
[41] G. Steward, Introduction to Matrix Computations, Academic Press, New York, 1973.