Connectivity Oblivious Merging of Triangulations

Luis F. Silva*, Luiz F. Scheidegger* Tiago Etiene† João L. D. Comba* Luis G. Nonato‡ Cláudio T. Silva§
*UFRGS, Brazil
†University of Utah, United States
‡USP-ICMC-São Carlos, Brazil
§Poly-NYU, United States

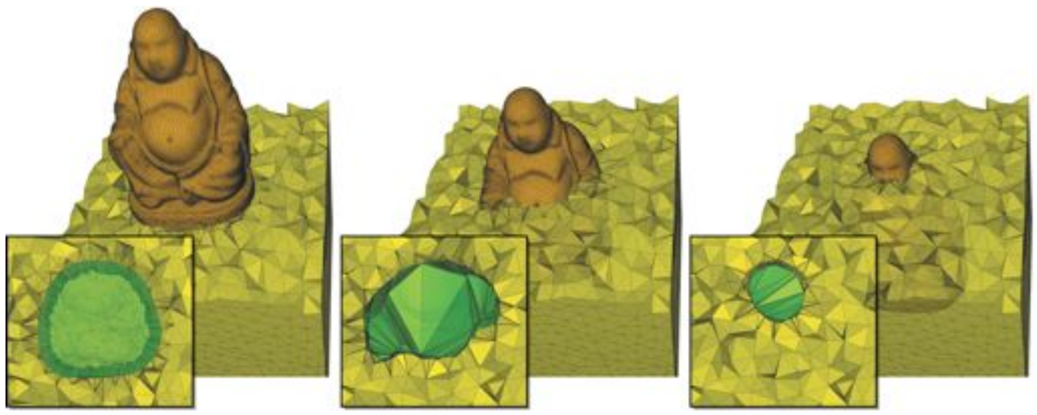


Fig. 1: *Merging a buddha tetrahedral mesh with a background grid. Our technique is able to handle meshes with distinct levels of refinement - observe how the internal tetrahedra of the buddha have not been refined.*

***Abstract*—Simplicial meshes are extremely useful as discrete approximations of continuous spaces in numerical simulations. In some applications, however, meshes need to be modified over time. Mesh update operations are often expensive and brittle, which tends to make the numerical simulations unstable. In this paper we propose an alternative technique for updating simplicial meshes that undergo geometric and topological changes. We exploit the property that a Weighted Delaunay Triangulation (WDT) can be used to implicitly define the connectivity of a mesh. Instead of explicitly maintaining connectivity information, we simply keep a collection of weights associated with each vertex. This approach allows for a simple way to merge triangulations, which we illustrate with examples in 2D and 3D.**

***Keywords*-Power diagram; Weighted Delaunay, Triangulations.**

I. Introduction

Simplicial meshes (triangulations) are the preferred data representation for techniques in computer graphics, visualization, and geometric computing. Their ability to represent irregular and adaptive domains is of particular importance in some applications. While triangulations are effective for *static* scenes, they are not so straightforward to use in *dynamic* settings. The main drawback in these situations is the fact that updating mesh connectivity during geometrical operations requires consistent and correct adjacency and incidence relations between simplicial elements. Much has been written about involved mechanisms for checking and repairing dynamic mesh structures (e.g., kinetic data structures [1]).

Dynamic updates of connectivity are particularly challenging for applications that involve merging of multiple meshes, or allow meshes to move over time. Recently, these challenges have been one of the drivers of the development of point-based (hybrid) approaches [2]. An alternative solution to this problem involves focusing only on triangulations whose connectivity can be implicitly derived from geometric constraints [3], [4], [5], such as Delaunay Triangulations (DT) [6]. However, these approaches are rather limited, since in some cases the underlying meshes do not comply with the required geometric constraints, thus falling back to the case where one needs to explicitly manage mesh connectivity [7].

Our proposal builds on the idea of implicit connectivity, based on a generalization of Delaunay Triangulations called Weighted Delaunay Triangulations (WDT). A property of DTs is the fact that all their vertices can be lifted onto a convex polyhedron in one extra dimension (also known as the *lifting property* [6], [8]). The same is not true for non-Delaunay meshes, however. WDTs are a generalization of DTs where a constant value is associated with each vertex in the mesh. This value modifies the height of the lifted vertices, thus moving them out of the lifted polyhedron. The main advantage of WDTs is their ability to represent a wider collection of meshes. Still, not every triangulation can be represented as a WDT. A key feature of our work is to use a subdivision scheme to turn any mesh into a WDT through the addition of extra vertices.

We build on previous work by Cignoni and De Floriani [9], who proposed a technique for computing a WDT from a given mesh by modeling the weight constraints as a linear program.

Unfortunately, this can not be done for all meshes, as the linear program might not have a solution in some cases. Instead, we apply a breadth-first traversal of the mesh that locally enforces convexity constraints on the lifting construction, together with a subdivision mechanism, which ensures a resulting WDT that conforms with the original mesh (more details in Section IV).

We present a general way to handle the lifting construction, which allows us to define different merging techniques. We also describe two approaches to merge WDTs based on different ways of lifting the polyhedron of each mesh. We also propose an adaptive scheme that maintains the geometric quality of simplices when merging meshes of distinct refinement levels. (see Section V-C). Figure 1 shows the merging of a tetrahedralization of the Buddha mesh with a background grid. We intentionally left the interior of the model hollow to highlight our technique's support of meshes with different levels of refinement.

Main Contributions: The main contributions of this work are a new approach for computing a WDT from an arbitrary simplicial mesh and a simple technique to merge multiple triangulations. We also describe a mechanism to smooth the transition between merged meshes with different refinement levels. In summary, the main features of our work are:

- *Simplicity:* The weight computation and merging schemes are all based on simple geometric constructions which do not demand complex data-structures;
- *Generality:* Our technique can automatically merge multiple triangulations with distinct mesh resolutions and arbitrary topology, while still ensuring consistent and well-defined connectivity;
- *Preservability:* Mesh features tend to be preserved during the merging process. Specifically, our approach easily preserves the anisotropy of a mesh.

We implemented our proposal and performed a number of experiments to validate the technique. In particular, we show examples that illustrate the merging of 2D and 3D meshes.

II. Related Work

Connectivity oblivious techniques for manipulation of simplicial meshes can be divided into two main categories: methods that reproduce an existing triangulation from geometric information, and methods that build a triangulation based on the constraints given by curves and/or surfaces defined in the domain of interest. In both cases, most methods rely on the properties of DTs and WDTs to avoid explicitly handling the mesh connectivity. These techniques have solid theoretical [8], [10], [6] and computational [11], [12] foundations. In particular, [6], [8]) are good starting references for WDT.

Techniques in the first category construct a WDT corresponding to a given mesh by computing a valid set of weights for its vertices. The advantage of this approach is that, once the weights are computed, connectivity information can be discarded. The original triangulation can then be rebuilt algorithmically from vertex coordinates and weights. One way to compute a valid set of weights is through a linear programming problem, which looks for an optimal weight distribution. Cignoni and De Floriani [9] use this formulation to perform depth sorting of unstructured simplicial meshes, without storing connectivity information. Balaven et al. [13] also use a linear programming approach to solve a problem similar to ours (see section V). However, their solution has the limitation of only dealing with meshes that can be directly represented as WDTs. When this is not the case, they resort to explicit management of connectivity information. The closest to our approach is the proposal by Pires et al. [14], where WDTs are used to compute morphing and simplification of meshes. Their formulation also relies on a linear program formulation for weight computation. On the other hand, our approach addresses the problem of weight computation in a different way, avoiding the linear programming formulation altogether. Since we allow small changes in the original mesh, our technique is always able to compute a valid set of weights that can be used to recover the given triangulation.

The second category of techniques is much more numerous, and an exhaustive description of it is beyond the scope of this paper. In general, these methods are concerned with either object modeling or domain decomposition for numerical simulation. For object modeling, the focus is on generating a triangulation that approximates a given model, using geometric certificates to filter undesirable simplices. Examples of connectivity oblivious object modeling techniques include Delaunay-based surface reconstruction from point clouds [15] and Delaunay-based isosurfacing methods [16], [17]. Domain decomposition techniques are designed to ensure that output meshes comply with constraints given by curves and/or surfaces defined inside the domain of interest. Enforcing the quality of simplicial elements is always a concern in these techniques. Delaunay mesh refinement [18], sliver removal [4], and Delaunay updates for moving meshes [7] are examples of connectivity oblivious domain decomposition techniques. The methodology we propose can be seen as a mechanism to extend mesh generation and object modeling techniques, since most operations can be implemented within our framework.

III. Background

In this section we introduce the basic concepts that are essential to the remainder of the paper, including fundamental ideas regarding regular and WDTs in Euclidean spaces. A more complete and general description of these geometrical entities can be found in [6], [19].

A. Regular and Weighted Delaunay Triangulations

Let $\mathcal{T}$ be a d-dimensional triangulation in $\mathbb{E}^d$, and $V = \{v_1, \dots, v_n\}$, $T = \{t_1, \dots, t_m\}$ be the set of vertices and d-simplices of $\mathcal{T}$ (d-simplices are triangles in $\mathbb{E}^2$ and tetrahedra in $\mathbb{E}^3$). The triangulation $\mathcal{T}$ is said to be *regular* if and only if there exists a set of points $V^+ = \{v_1^+, \dots, v_n^+\} \in \mathbb{E}^{d+1}$ such that V^+ is a subset of the vertices of a convex polyhedron $\mathcal{P}$ where, for all i, v_i^+ projects orthogonally on v_i, and the projection of downward d-faces of $\mathcal{P}$ is T (a d-face t is a *downward face* if $n_t \cdot e_{d+1} < 0$, where $e_{d+1} = (0, \dots, 0, 1) \in \mathbb{E}^{d+1}$ and n_t is a vector normal to

t). In other words, a d-dimensional regular triangulation is the vertical projection of the "lower side" of some $(d+1)$-dimensional convex polyhedron in $\mathbb{E}^{d+1}$.

The WDT is a well-known regular triangulation. Given a set of points $V = \{v_1, \ldots, v_n\}$ in $\mathbb{E}^d$ and corresponding scalar weights $W = \{w_1, \ldots, w_n\}$, the WDT is obtained by projecting downward faces of the so-called *lifted polyhedron* (a polytope, to be more precise), defined as the convex hull of points $V^+ = \{v_1^+, \ldots, v_n^+\} \subset \mathbb{E}^{d+1}$, where

$$v_i^+ = (v_{i_{x_1}}, \ldots, v_{i_{x_d}}, v_{i_{x_1}}^2 + \cdots + v_{i_{x_d}}^2 - w_i) \qquad (1)$$

is a *lifted vertex* and $\{v_{i_{x_1}}, ..., v_{i_{x_d}}\}$ are the Cartesian coordinates of v_i. It is not difficult to observe how a lifted vertex v_i^+ may be inside the convex hull of V^+ (suppose, for example, that the weight w_i is much smaller than the weights of v_i's neighbors). This means v_i^+ will not be in any downward face of the lifted polyhedron, leaving v_i absent from the WDT of V. Absent vertices are called *redundant* and will appear naturally in the merging scheme described in Section V.

It is worth noting that the lifting map defined for weighted vertices can be extended to simplices of a triangulation $\mathcal{T}$ in a straightforward way. For example, the lifting of a k-simplex r with vertices $v_{r_0}, \ldots, v_{r_k}$ is the k-simplex r^+ in $\mathbb{E}^{d+1}$ whose vertices are $v_{r_0}^+, \ldots, v_{r_k}^+$, with coordinates given by (1).

B. Local Convexity

Let t_i and t_j be two adjacent d-simplices sharing a common $(d-1)$-face r_{ij} (r_{ij} will be an edge in $\mathbb{E}^2$ and a triangle in $\mathbb{E}^3$) and t_i^+, t_j^+ be the lifted simplices of t_i and t_j. The $(d-1)$-face r_{ij}^+ is said to be *locally convex* if the $(d+1)$-simplex generated from the union of vertices in t_i^+ and t_j^+ lies on the upper half-space of each hyperplane containing t_i^+ and t_j^+. A polyhedron in $\mathbb{E}^{d+1}$ is called locally convex if all $(d-1)$-faces shared by two d-simplices are locally convex.

Local convexity is a necessary, but not sufficient, condition to ensure global convexity of a polyhedron [20]. However, in the particular case of a polyhedron generated by lifting a convex triangulation, local convexity leads to global convexity (see [21] for details) and, as remarked by Balaven et al. [13], this property has widely been used as a mechanism to verify whether a given triangulation is regular. As we show in the next section, the association between lifting and local convexity comprises the core of our approach.

It is important to highlight that not every triangulation is regular. In [12], it is presented seven examples of meshes that are not regular. Because of this, it may be impossible to build a convex polyhedron whose projection matches a given triangulation, and therefore any lifting of this triangulation will contain at least one non-locally convex $(d-1)$-simplex.

IV. WDT Computation and Conformal Refinement

Given a triangulation $\mathcal{T}$ in $\mathbb{E}^d$, we compute a set of weights for its vertices to represent $\mathcal{T}$ as a WDT. We rely on a geometric construction that exploits the lifting and local convexity properties in $\mathbb{E}^{d+1}$ to compute the weights. A triangulation conforming to the original mesh can be reconstructed by an algorithm that computes a WDT from vertex coordinates and associated weights (see, for example, [12]). Therefore, incidence and adjacency relations between simplices are implicit in the weighted vertices, and are not stored explicitly.

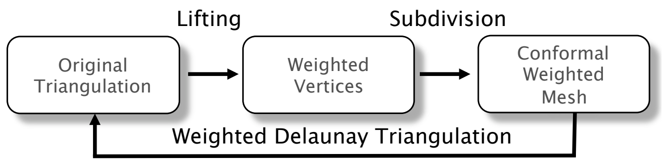


Fig. 2: *WDT computation: original triangulation is traversed in breadth-first manner to assign weights to vertices using the lifting construction. A subdivision step follows to ensure the weighted mesh conforms with the original triangulation.*

Conventional approaches to weight computation look for a solution to a set of linear inequalities, or try to solve a linear programming problem [13], which may not have a solution. We, on the other hand, use a deterministic procedure that builds a polyhedron in $\mathbb{E}^{d+1}$ that is locally convex as much as possible. This step generates a preliminary weight assignment that might not enforce local convexity to some $(d-1)$-simplices. We then employ a subdivision process to handle these simplices, thus ensuring that a convex polyhedron is generated as output. As a result, the original triangulation can be rebuilt from the weighted vertices, even though a few simplices of $\mathcal{T}$ may appear subdivided after the reconstruction. The diagram in Figure 2 provides an overview of our approach.

A. Computing Weights

Let t_i be a d-simplex of a triangulation $\mathcal{T}$ in $\mathbb{E}^d$ and t_j be another d-simplex in $\mathcal{T}$ sharing a $(d-1)$-face r_{ij} with t_i. Suppose that weights have already been assigned to vertices of t_i, such that one can compute the hyperplane π_i in $\mathbb{E}^{d+1}$ containing t_i^+. In other words, π_i is the *support hyperplane* of t_i^+. Let v be the vertex of t_j opposite to r_{ij} and $\tilde{v}$ be the vertical projection of v onto π_i, as illustrated in Figure 3. Notice that a small vertical perturbation of $\tilde{v}$ toward the positive half-space of π_i, denoted as v^+, makes the simplex r_{ij}^+ locally convex with respect to t_i^+ and $t_j^+ = r_{ij}^+ \cup v^+$. Therefore, if the coordinates of $\tilde{v}$ are $(v_{x_1}, \ldots, v_{x_d}, \tilde{v}_{x_{d+1}})$, we use equation (1) to define the weight w_v of the vertex v of the vertex v as:

$$w_v = v_{x_1}^2 + \ldots + v_{x_d}^2 - (\tilde{v}_{x_{d+1}} + \epsilon) \qquad (2)$$

where ϵ is a small vertical perturbation of $\tilde{v}$ that makes r_{ij}^+ locally convex with regard to t_i^+ and t_j^+.

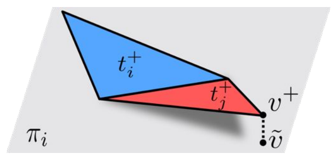


Fig. 3: *Vertex lifting to ensure local convexity.*

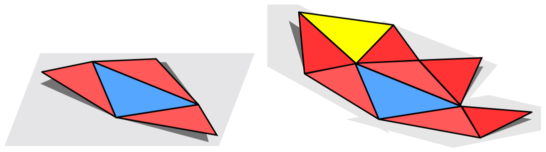


Fig. 4: *Simplices are lifted in a breadth-first fashion. Red simplices are lifted to ensuring local convexity, but this can not be ensured for the edges of the yellow simplex.*

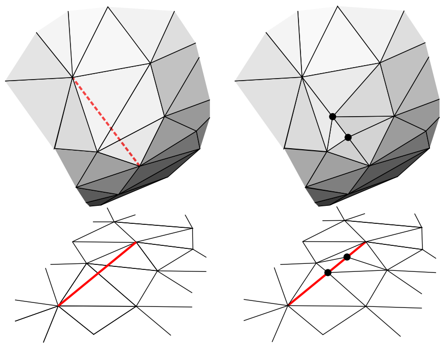


Fig. 5: *Inserting absent edges. A red edge absent from the final triangulation (left), is included using several smaller edges (right), with a subdivision process that generates new vertices with proper weight assignments.*

By repeating the construction above for all neighboring simplices of each lifted d-simplex (red triangles in Figure 4), we can ensure that each lifted $(d-1)$-simplex is locally convex. We initialize our algorithm by selecting an arbitrary d-simplex (a triangle in $\mathbb{E}^2$ or a tetrahedron in $\mathbb{E}^3$) from the input triangulation, and setting its vertices' weights to zero.

Although weights are assigned consistently to vertices of $\mathcal{T}$, some $(d-1)$-simplices of the original triangulation are not handled during the weight computation process described above (see the yellow triangle in Figure 4). This is not an issue for most $(d-1)$-simplices, which are locally convex naturally, and appear in the reconstructed triangulation despite not being processed. We describe next our handling of simplices that are not locally convex after the lifting is complete.

B. Enforcing Conformal Meshes

Absent simplices can be forced to appear in the weighted triangulation by using a subdivision process, also employed in Delaunay-based mesh generation and modeling techniques [22]. Such subdivision strategies ensure that the reconstructed weighted triangulation conforms with the original triangulation. Formally, given a triangulation $\mathcal{T}$ in $\mathbb{E}^d$, the subdivision ensures that for each k-simplex r of $\mathcal{T}$ there exists k-simplices $r_{w_1}, \ldots, r_{w_s}$, $s \geq 1$ in $\mathcal{T}_w$ (the reconstructed weighted triangulation) such that $|r| = |r_{w_1} \cup \cdots \cup r_{w_s}|$, where $|r|$ is the underlying space of r.

Suppose that an edge $e \in \mathcal{T}$ is absent from $\mathcal{T}_w$, where $\mathcal{T}_w$ is the weighted triangulation obtained as described before. The subdivision step consists of splitting e at its midpoint v_e, and assigning a weight to v_e to ensure that it appears in the weighted triangulation. As shown in the left of Figure 5, the absent edge e intersects a set T_{w_e} of d-simplices in $\mathcal{T}_w$. To keep changes as local as possible, we ensure that the weight of v_e only alters the simplices of T_{w_e}, and keep the remaining triangulation unchanged. This property can be satisfied by computing the weight of v_e as follows: let t_{v_e} be the d-simplex of $\mathcal{T}_w$ containing v_e and $t^+_{v_e}$ be the lifting of t_{v_e} in $\mathbb{E}^{d+1}$. We call l_{v_e} the vertical line through v_e, and compute the intersection point $\tilde{v}_{e_1}$ between l_{v_e} and the support hyperplane of $t^+_{v_e}$, as well as the highest intersection $\tilde{v}_{e_2}$ of l_{v_e} with the hyperplanes supporting the lifted d-simplices in $\tilde{\mathcal{T}}_w - T_{w_e}$.

It is easy to see that any choice of weight assignment that places v^+_e strictly between $\tilde{v}_{e_1}$ and $\tilde{v}_{e_2}$ ensures that v_e will appear in the weighted triangulation (v^+_e will be underneath the lifted convex hull). This approach only modifies the simplices in T_{w_e}, since v^+_e is above the hyperplanes supporting the simplices in $\tilde{\mathcal{T}}_w - T_{w_e}$. In our current implementation, we set a weight that places v^+_e halfway between $\tilde{v}_{e_1}$ and $\tilde{v}_{e_2}$. Although this procedure adds v_e to the weighted triangulation, it does not ensure that each sub-segment of e will appear in the triangulation. If a sub-segment still does not appear, we recursively apply the same procedure. Since the lifting of each new vertex is placed below existing simplices, the subdivision process finishes in a finite number of steps. In practice, few absent simplices demand more than one or two iterations.

The procedure described above for recovering absent edges can be extended to enforce triangular faces in 3D complexes. The centroid of each missing triangular face is inserted and positioned exactly as was done for absent edges. In fact, in the 3D case, we first recover all missing edges before splitting absent faces, since most missing triangles naturally appear after the edges are recovered.

V. Merging Weighted Delaunay Triangulations

One of the main advantages of a WDT is the fact that incidence and adjacency relations among simplices are automatically resolved, thus avoiding the need for connectivity manipulation. This benefit can be explored in applications involving the interaction between two or more meshes, where the guarantee of consistent connectivity between the meshes is usually a complex and painful task. Below, we describe two techniques that allow WDTs to be merged using a lifted polyhedra construction, designed to keep changes as local as possible and only modify a small neighborhood of the merged region. Since meshes are often represented in different levels of detail or refinement, merging results are prone to generate badly shaped simplicial elements. We address this issue with a simple fringe mechanism.

A. Convexification-based Merging

We call the first proposal *convexification-based* merging. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two triangulations and suppose that $\mathcal{T}_2$ must be merged with $\mathcal{T}_1$ while keeping $\mathcal{T}_1$ as unchanged as possible (i.e. we only want to affect a neighborhood surrounding $\mathcal{T}_2$). We position $\mathcal{T}_2$ inside $\mathcal{T}_1$ and lift the polyhedron $\mathcal{T}_2$ in such a way that $\mathcal{T}_2^+$ is squeezed between the lifted polyhedron $\mathcal{T}_1^+$

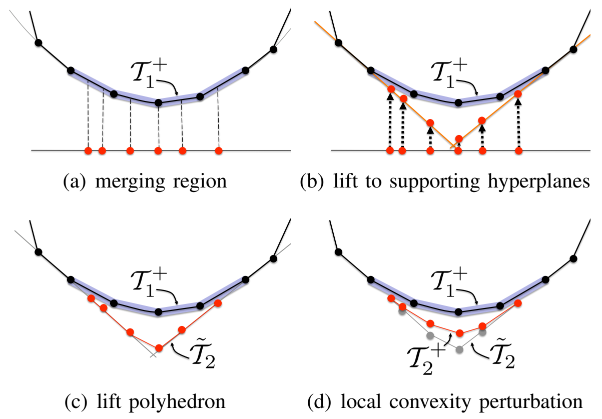


Fig. 6: *Convexification-based merging*

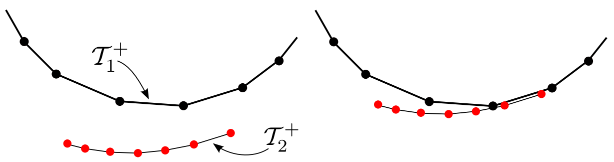


Fig. 7: *Translation-based merging.*

and the supporting hyperplanes for those simplices of $\mathcal{T}_1^+$ not affected by the merging (Figure 6).

The squeezing process is composed of three steps. First, the d-simplices of $S \subset \mathcal{T}_1$ that do not intersect $\mathcal{T}_2$ are identified (Figure 6.a) and their supporting hyperplanes computed (Figure 6.b). In the second step, we set the weight of each vertex v of $\mathcal{T}_2$ in such way that the vertex is placed at the highest intersection $\tilde{v}$ between the vertical line passing through v and the supporting hyperplanes of the d-simplices in S (Figure 6.b). This construction results in a polyhedron in $\mathbb{E}^{d+1}$, denoted by $\tilde{\mathcal{T}}_2$(Figure 6.c). The final step makes the $(d-1)$-simplices of $\tilde{\mathcal{T}}_2$ locally convex, using a subdivision procedure as before. Starting from the d-simplices $\tilde{t}_i^+ \in \tilde{\mathcal{T}}_2$ with at least one face on the boundary of $\mathcal{T}_2$, we apply a small vertical perturbation on each vertex opposite to $\tilde{t}_i^+$, to position it above the supporting hyperplane of $\tilde{t}_i^+$ and below $\mathcal{T}_1^+$. We repeat this procedure recursively, moving vertices towards the interior of $\tilde{\mathcal{T}}_2$ (Figure 6.d). Finally, we subdivide absent simplices to force them to appear. Since vertices of $\mathcal{T}_1$ inside $\mathcal{T}_2$ are placed in the interior of the convex hull of $\mathcal{T}_1^+ \cup \mathcal{T}_2^+$, they become redundant and do not appear in the merged triangulation.

Although the convexification-based approach ensures that the only simplices of $\mathcal{T}_1$ affected by the merging are the ones in S, it is also prone to discard more elements of $\mathcal{T}_2$. In contrast to the approach of Section IV-A that propagates a front while computing weights, here we lift $\mathcal{T}_2$ by propagating different fronts simultaneously from the elements at the boundary of the merging region. Therefore, no guarantee can be given with respect to the local convexity for simplices at the union of two or more fronts, and non-locally convex faces can appear.

B. Translation-based Merging

We implemented a second merging scheme, called *translation-based merging*, to reduce the number of missing elements in $\mathcal{T}_2$. This approach is prone to change more simplices in $\mathcal{T}_1$ than the previous one, but it also reduces the number of missing elements of $\mathcal{T}_2$ considerably. We apply the lifting scheme described in Section IV-A to $\mathcal{T}_1$ and $\mathcal{T}_2$ independently, but using a vertical displacement for $\mathcal{T}_1$ twice as large as the one used for $\mathcal{T}_2$. More specifically, the value of ϵ in equation (2) is halved when lifting $\mathcal{T}_2$, thus making the lifted polyhedron $\mathcal{T}_2^+$ less curved than $\mathcal{T}_1^+$. Once weights have been assigned to vertices in $\mathcal{T}_1$ and $\mathcal{T}_2$, we displace $\mathcal{T}_2^+$ vertically (adding a constant to all weights) until it "touches" $\mathcal{T}_1^+$ from below, as illustrated in Figure 7. As $\mathcal{T}_2^+$ is less curved than $\mathcal{T}_1^+$, their support hyperplanes tend to be below $\mathcal{T}_1^+$, thus keeping $\mathcal{T}_2$ unchanged after the merging process (including simplices subdivided during the weight assignment step). Although new subdivisions can appear in $\mathcal{T}_2$, and $\mathcal{T}_1$ can be affected in a wider neighborhood of $\mathcal{T}_2$, this merging method behaves well in practice, as shown in Section VI.

C. Fringe

Badly shaped simplices are likely to appear during the merging process, especially when meshes have distinct refinement levels. In order to address this problem, we create an intermediate triangulation, called *fringe*, that makes a smoother transition between meshes to be merged together.

The fringing process works as follows: we initially compute an estimate of local density. We assign an average edge length to each vertex, and normalize these values to make the largest average equal to 1. We then define a regular grid whose resolution is given by the smallest edge present in the scene, and place it around the inner mesh. The density estimates for each vertex serve as boundary conditions on this grid, and we run a Laplacian solver to smoothly transition between densities on the inside and on the border of the grid.

The solution of the Laplacian smoothing solver can be seen as the probability that a vertex on the regular grid should be present in a transition mesh. Therefore, we selectively maintain vertices of the grid based on their density estimates. Finally,

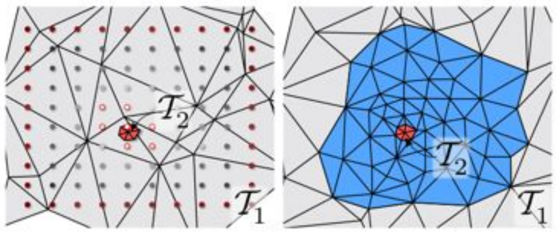
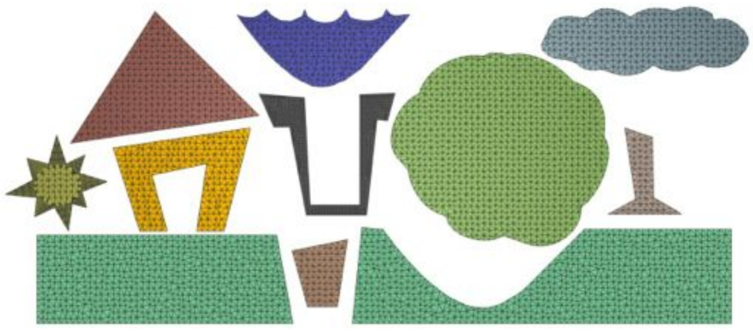
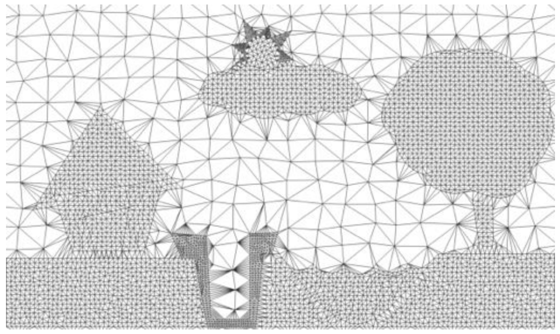


Fig. 8: *Fringing. On the left, two meshes are merged, along with the regular grid containing the local density estimates for the meshes. Red circles mark boundary conditions on the Laplacian Smoothing Solver. On the right, the smaller mesh is merged to a fringe (in blue) that makes a graceful transition between the resolution of the internal and external meshes.*


(a) 2D meshes of the cartoon animation


(b) Merging of all meshes in a given frame

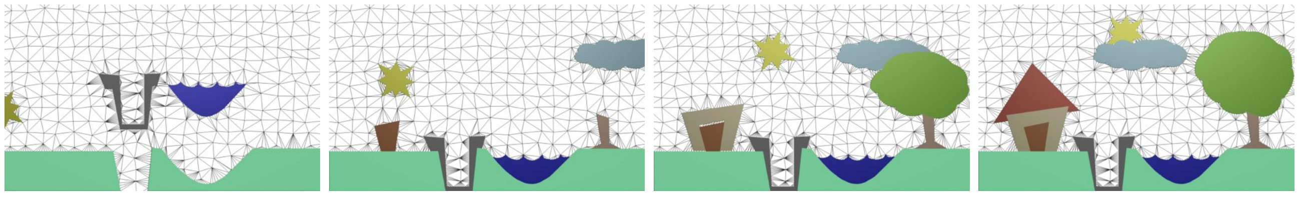

(c) Four snapshots of the animation sequence

Fig. 9: *2D Cartoon Animation illustrating merging with several dynamic meshes. See video in supplemental material.*

we construct the fringe by generating a Delaunay Triangulation of the points that were maintained. We also apply one iteration of Laplacian smoothing solver to the fringe to make its simplices better shaped. Once the fringe is constructed, the merging is computed in two steps: first, we merge the internal mesh with the fringe, and then merge the result with the outer mesh (Figure 8).

VI. Experimental Results and Applications

We implemented the techniques described above using the CGAL computational geometry library. We used CGAL's support for WDT computation. We computed vertex coordinates and weights computed with our method, and CGAL to compute the WDT. This has shown to be robust in the tests we performed. A series of merging examples is given below containing several triangulations of different characteristics, including 2D and 3D meshes. In all experiments conducted, we set $\epsilon = 10^{-4}$ (see equation 2 for details).

A. Merging 2D meshes

The ability and flexibility of merging triangulations is illustrated first with a physically-based cartoon animation of 2D meshes. The animation is composed of a scene with meshes that move inside a background triangulation. The background mesh performs collision detection and simplifies object placement, since we can compute the adjacency between distinct objects by checking whether edges in the background grid connect them. Figures 9(a) and 9(b) show the meshes and the merging result using the translation-based scheme.

Figure 9(c) has four intermediate frames illustrating the capability of our technique to handle multiple meshes simultaneously. In our implementation the elastic deformation of each object was pre-computed. The order in which objects are merged naturally solves the problem of front-to-back alignment. In this example, we prioritize the mesh of the last object merged, thus placing it in front of all the others.

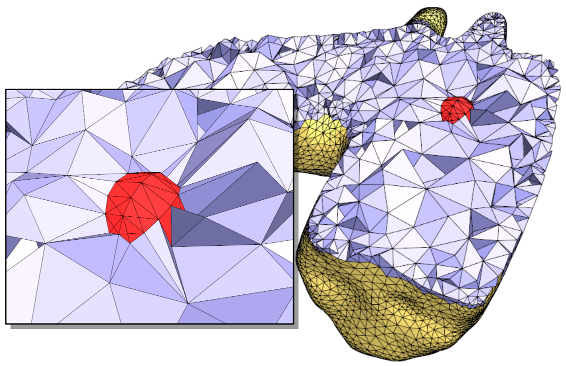
As algorithms to compute the WDT always generate a triangulation for the convex hull of the weighted points, we have to handle concavities explicitly. Vertices in the background grid inside a concavity of a merged mesh can be forced to appear by adjusting their weights. This adjustment makes the vertices' lifted counterparts lie below the convex polyhedron of all merged meshes, above their supporting hyperplanes. In this example, we only handle missing edges at the boundary of objects, while not forcing missing edges in the interior of objects to appear. Therefore, mesh flips can be observed.

B. Merging 3D meshes

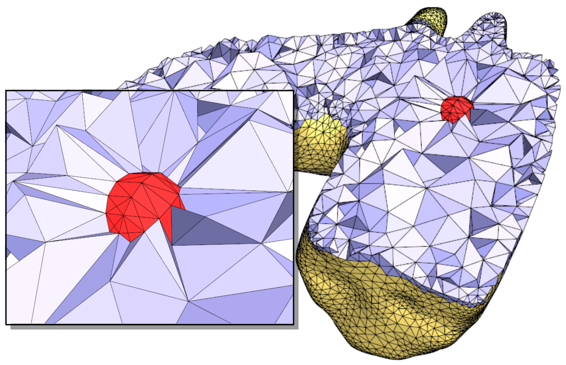
The second set of experiments shows the interaction between tetrahedral meshes. Due to occlusion, screenshots of the merging result in 3D hardly convey the strength of our connectivity oblivious technique. Therefore, we present renderings of cutaway sections of the models. In addition, we first validated merging results without fringing, since we wanted to stress the impact at the tetrahedra closer to the merging region.

The first experiment is composed of a moving tetrahedral sphere passing through a mesh of the aorta artery. Figures 10(a) and 10(b) show the sphere merged inside the aorta using both the convexification and translation-based merging schemes. Even though the convexification-based approach impacts a smaller neighborhood around the merging region, more absent simplices tend to be generated with this technique.

In Figure 10(b) we observe that some vertices in the neighborhood of the sphere were removed (became redundant) when using translation-based merging. However, no missing simplices were identified in the translation-based approach,



(a) Convexification-based merging



(b) Translation-based merging

Fig. 10: *Merging the aorta artery tetrahedral mesh with a tetrahedral decomposition of a sphere. a) Convexification-based merging, 4.63% missing edges; b) Translation-based merging, no absent edges. See video in supplemental material.*

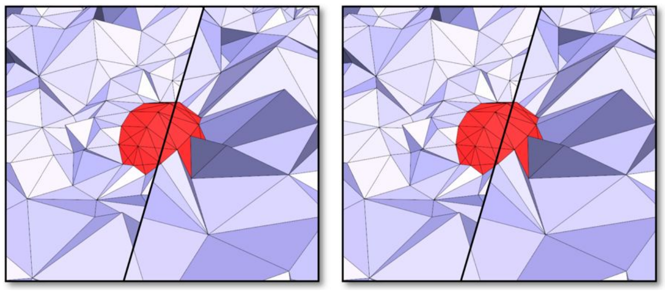


Fig. 11: *Comparison between fringe merging (left) and convexification-based merging (right). The fringe ensures a smoother transition between the two merged meshes, thus minimizing the occurrence of elongated, low-quality tetrahedra.*

while 4.63% of missing simplices were found and subdivided using the convexification-based merging (all inside the sphere) Figure 11 compares the result of convexification-based merging in Figure 10 to the result obtained using fringing, which allows a smoother transition between the aorta and the sphere.

Table I lists the number of vertices in each mesh, as well as the time to perform weight computation and the triangulation. We can observe that timings for weight computation are negligible, the most expensive operation is the triangulation.

Dataset	Vertices	Weight Computation	Triangulation
Aorta	9529	45.05	1259.52
Sphere	58	0.094	12.28
Grid 1	50480	305.15	6633.47
Rocker	10713	53.24	1662.97
Rod	3098	10.24	870.40
Valve	3857	16.38	882.68
Grid 2	8371	30.72	744.14
Buddha	27975	178.17	5230.59

TABLE I: *Mesh data and WDT computation (all times in ms)*

Merging Meshes	Convexification-based	Translation-based
Aorta-Sphere	53.24	63.48
Grid 1-Rocker	33062.91	38082.17
Rocker-Arm	528.38	503.80
Valve-Rod	223.23	196.69
Grid 2-Buddha	61526.01	665647.87

TABLE II: *Merging Times (all times in ms)*

Table II shows the merging time in both approaches. In the second example, three merges are used to compute the result. Both merging methods produce similar performance results.

Figure 12 illustrates how our technique handles large and complex tretrahedral meshes with arbitrary topology. Figure 12(a) shows the tetrahedral meshes (only boundary faces are shown) of three mechanical pieces merged inside a cubic background grid. Notice from the close-ups that the anisotropy of the gray meshes has been preserved after merging. In fact, no missing simplices (thus no new vertices) have appeared after merging these models using the translation-based scheme.

VII. Discussion and Limitations

A remarkable aspect of our approach is that, despite its simplicity, it is able to handle and merge complex simplicial meshes without managing connectivity explicitly, as is usually done with conventional Delaunay methods. However, it is important to point out that there exist significant differences between our approach and Delaunay-based schemes. Our technique starts from a set of triangulations, whereas Delaunay-based methods build a triangulation from curves and surfaces constraining the domain of interest. The difference between the two methodologies becomes more evident when dealing with anisotropic meshes. Although anisotropic versions of Delaunay Triangulations and Constrained Delaunay Triangulations have been used to handle anisotropic meshes [23], such techniques demand either tensor estimation to define the anisotropy or an explicit manipulation of constraints, which can become as complex as the connectivity management during merging operations. In fact, our approach can be seen as a first step toward a new framework for connectivity oblivious mesh representation using the idea of regular triangulations.

In some applications, the translation-based merging scheme may affect a wide neighborhood around the merging region, making vertices redundant. Although one can always reinstate missing vertices by vertically displacing then beneath the convex hull of lifted points, checking for redundancy can become

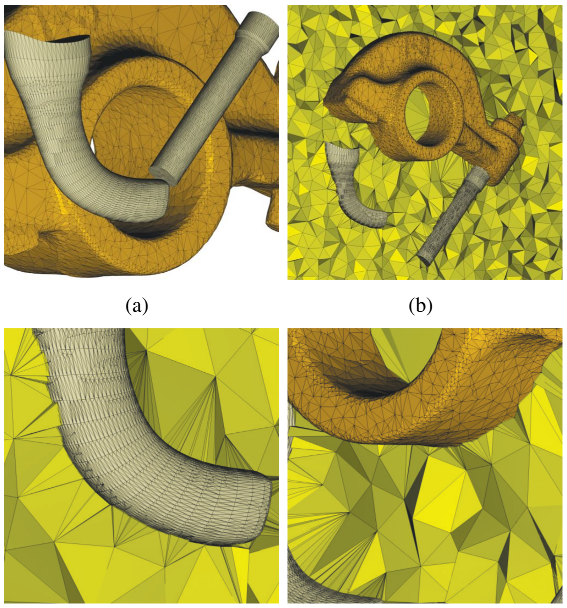

(a) (b) (c) (d)

Fig. 12: *Merging in 3D: (a) tetrahedral meshes to be merged (boundary surface is shown), (b) mechanical pieces merged in the background grid; (c) and (d) closeup view show the original meshes were preserved after merging.*

costly. Plane projection plus convexification can indeed ensure local mesh updates for the background grid. However, the number of internal missing simplices exceeded 10% in one of our experiments (our worst case), which can be unacceptable in applications where internal meshes must be preserved. This large number of absent elements, in some cases, is due to numerical instabilities, as the cavity between the convex hull and the support planes can become too narrow. This makes it numerically difficult to convexify the lifted polyhedron with vertical perturbations. If an excessively large vertical perturbation is applied to a vertex, numerical predicates can “see” adjacent simplices as coplanar, violating what is known as the *general position hypothesis*. This makes it difficult to predict which vertices will be seen as coplanar.

VIII. Conclusions

In this work, we proposed methods, based on weighted Delaunay Triangulations, to merge simplicial complexes without the need for explicit management of connectivity information. We also presented a new algorithm to compute these weights based on breadth-first traversal of the mesh. Merging results are satisfactory, in particular with respect to the algorithm's stability and locality of mesh updates.

In the future, we would like to revisit the weight assignment technique, to better enforce local convexity of the lifted polyhedron. Another interesting avenue for future research is to combine our framework with numerical simulations. Since our merging technique generates high quality simplices, we believe it may be coupled with existing numerical models.

Acknowledgment

This work was supported in part by the National Science Foundation (CCF-08560, CCF-0702817, CNS-0751152, CNS-1153503, IIS-0844572, IIS-0904631, IIS-0906379, IIS-1153728), Department of Energy, FAPESP, CNPq (309483/2011-5, 131322/2008-7, 491034/2008-3), IBM Faculty Awards and NVIDIA Fellowships.

References

[1] L. Guibas, “Kinetic data structures,” in *Handbook of Data Structures and Applications*, 2004.
[2] M. Gross and H. Pfister, Eds., *Point-Based Graphics*, 2007.
[3] J. Tournois, C. Wormser, P. Alliez, and M. Desbrun, “Interleaving Delaunay refinement and optimization for practical isotropic tetrahedron mesh generation,” *ACM Trans. Graph.*, vol. 28, no. 3, pp. 1–9, 2009.
[4] S.-W. Cheng, T. K. Dey, H. Edelsbrunner, M. A. Facello, and S.-H. Teng, “Sliver exudation,” in *SCG '99: Proceedings of the fifteenth annual symposium on Computational geometry*, 1999, pp. 1–13.
[5] J. Tournois, R. Srinivasan, and P. Alliez, “Perturbing slivers in 3d Delaunay meshes,” in *International Meshing Roundtable 2009*, 2009.
[6] H. Edelsbrunner and R. Seidel, “Voronoi diagrams and arrangements,” *Discrete Comput. Geom.*, vol. 1, pp. 25–44, 1986.
[7] P. M. M. de Castro, J. Tournois, P. Alliez, and O. Devillers, “Filtering relocations on a Delaunay triangulation,” *Computer Graphics Forum*, vol. 28, no. 5, pp. 1465–1474, 2009.
[8] F. Aurenhammer, “Power diagrams: Properties, algorithms and applications,” *SIAM J. Comput.*, vol. 16, no. 1, pp. 78–96, 1987.
[9] P. Cignoni and L. De Floriani, “Power diagram depth sorting,” in *Proceedings of the 10th Canadian Conference on Computational Geometry*, M. Soss, Ed., 1998, pp. 88–89.
[10] F. Aurenhammer and H. Imai, “Geometric relations among Voronoi diagrams,” in *4th Annual Symposium on Theoretical Aspects of Computer Sciences on STACS 87*, 1987, pp. 53–65.
[11] M. Vigo, N. Pla, and J. Cotrina, “Regular triangulations of dynamic sets of points,” *Comput. Aided Geom. Des.*, vol. 19, no. 2, pp. 127–149, 2002.
[12] H. Edelsbrunner and S. Shah, “Incremental topological flipping works for regular triangulations,” *Algorithmica*, vol. 15, pp. 223–241, 1996.
[13] S. Balaven, C. Bennis, J.-D. Boissonnat, and M. Yvinec, “Conforming orthogonal meshes,” in *IMR*, 2002, pp. 219–227.
[14] F. B. Pires, C. A. Dietrich, J. L. D. Comba, and L. G. Nonato, “Mesh processing using on-the-fly connectivity reconstruction given by regular triangulations,” in *Proceedings of the 2010 23rd SIBGRAPI Conference on Graphics, Patterns and Images*, ser. SIBGRAPI '10, 2010, pp. 87–94.
[15] N. Amenta, S. Choi, and R. Kolluri, “The power crust,” *Computatinal Geometry*, vol. 19, pp. 127–153, 2001.
[16] S.-W. Cheng, T. K. Dey, E. A. Ramos, and T. Ray, “Sampling and meshing a surface with guaranteed topology and geometry,” in *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, 2004, pp. 280–289.
[17] J.-D. Boissonnat and S. Oudot, “Provably good sampling and meshing of surfaces.” *Graphical Models*, vol. 67, pp. 405–451, 2005.
[18] J. Shewchuk, “Delaunay refinement algorithms for triangular mesh generation,” *Computational Geometry: Theory and Applications*, vol. 22, no. 2-3, pp. 21–74, 2002.
[19] H. Edelsbrunner, *Geometry and Topology for Mesh Generation*, 2001.
[20] K. Mehlhorn, S. Ngher, T. Schilz, S. Schirra, M. Seel, R. Seidel, and C. Uhrig, “Checking geometric programs or verification of geometric structures,” in *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, 1996, pp. 159–165.
[21] O. Devillers, G. Liotta, F. Preparata, and R. Tamassia, “Checking the convexity of polytopes and the planarity of subdivisions,” *Computational Geometry*, vol. 11, pp. 187–208, 1998.
[22] L. G. Nonato, A. Cuadros-Vargas, R. Minghim, and M. Oliveira, “Beta-connection: Generating a family of models from planar cross sections,” *ACM Transactions on Graphics*, vol. 24, no. 4, pp. 1239–1258, 2005.
[23] H. Borouchaki, P. L. George, F. Hecht, P. Laug, and E. Saltel, “Delaunay mesh generation governed by metric specifications. part i. algorithms,” *Finite Elements in Analysis and Design*, vol. 25, pp. 61–83, 1997.