Spatio-Temporal Shape from Silhouette using Four-Dimensional Delaunay Meshing

Ehsan Aganj, Jean-Philippe Pons, Florent Ségonne and Renaud Keriven
WILLOW Team, CERTIS, École des ponts
Marne-la-Vallée, France
{aganj,pons,segonne,keriven}@certis.enpc.fr

Abstract

We propose a novel method for computing a four-dimensional (4D) representation of the spatio-temporal visual hull of a dynamic scene, based on an extension of a recent provably correct Delaunay meshing algorithm. By considering time as an additional dimension, our approach exploits seamlessly the time coherence between different frames to produce a compact and high-quality 4D mesh representation of the visual hull. The 3D visual hull at a given time instant is easily obtained by intersecting this 4D mesh with a temporal plane, thus enabling interpolation of objects' shape between consecutive frames. In addition, our approach offers easy and extensive control over the size and quality of the output mesh as well as over its associated reprojection error. Our numerical experiments demonstrate the effectiveness and flexibility of our approach for generating compact, high-quality, time-coherent visual hull representations from real silhouette image data.

1. Introduction

The problem of estimating the shape of an object or a scene from multiple views has received a considerable interest in computer vision. *Shape from silhouette* is one popular class of methods to solve this problem in an approximate but efficient and robust manner. Most of these techniques consist in computing the *visual hull*, which is the maximal volume consistent with a given set of silhouettes. One noticeable exception is the recent work of Franco *et al.* [14] which constructs smooth silhouette-consistent shapes strictly included in the visual hull.

The concept of visual hull was first introduced by Baumgart [3], it was reviewed by Laurentini [19], and it has been extensively studied since then. Visual hull algorithms fall into two main categories: *volume-based* approaches and *surface-based* approaches. However, for sake of completeness, let us also mention the image-based approach of Matusik *et al.* [24] that generates on-the-fly a view-dependent visual hull representation for the purpose of rendering.

Volume-based methods [13, 26, 29, 30] use a subdivision of space into elementary regions, typically voxels. These methods suffer from quantization and aliasing artefacts, as well as from a high memory cost. Moreover, when a final mesh representation is required, they must be complemented with an isocontour extraction algorithm, such as the *marching cubes* algorithm, introduced by Lorensen and Cline [22].Unfortunately, this technique produces unnecessarily large meshes (at least one triangle per boundary voxel) of very low quality (lots of skinny triangles). Frequently, the resulting meshes have to be regularized, optimized and decimated in order to obtain suitable representations in terms of compactness and quality, while simultaneously controlling the approximation accuracy and preserving some topological properties, such as the absence of self-intersections, which turns out to be a difficult task.

Surface-based methods [8, 12, 16, 20, 23] consist in directly building a mesh representation of the visual hull by computing the intersection of viewing cone surfaces associated with the silhouettes. It is generally admitted that these algorithms suffer from numerical instability in the case of noisy or mutually inconsistent silhouette data. Moreover, the output mesh has a very low quality, and typically develops degeneracies when the number of input views increases. For example, in the case of exact polyhedral intersection [12, 23], very small triangles uselessly augments the size of the visual hull representation. More generally, in these approaches, there is no easy way to control the size or the quality of the output mesh, because it is related in a very complex way to the chosen polygonization of image contours.

While many authors have focused on computing the visual hull in the case of static images, leading to several established techniques mentioned above, very little work has dealt with the case of dynamic scenes captured by multiple video sequences, from an actual spatio-temporal perspective, i.e. by going beyond independent frame-by-frame computations.

The most notable related work is that of Cheung *et al.* [9, 10] on computing visual hull across time. However, their purpose is quite different from ours. Their goal is to virtually increase the number of input silhouettes by registering many views of a rigidly-moving object in a common reference frame, in order to approximate the shape of the object more accurately. See also [31] for an early work in this direction. More related to our problem is the extension of this approach to articulated motion [9, 10]. Under the articulated assumption, their approach exploits temporal coherence to improve shape reconstruction while estimating the skeleton of the moving object. However, their approach is experimentally validated in the case of a single joint only. Hence it is not relevant to the case of complex human motion or non-rigid scenes.

In this paper, we propose a novel method to compute a four-dimensional (4D) representation of the spatio-temporal visual hull of a non-rigid dynamic scene. Our work builds on a recent provably correct Delaunay-based algorithm for meshing surfaces, from Boissonnat and Oudot [5, 6]. This algorithm is proven to terminate and to construct good-quality meshes, while offering bounds on the approximation accuracy of the original boundary and on the size of the output mesh. The refinement process is controlled by highly customizable quality criteria on triangular facets. A notable feature of this method is that the surface needs only to be known through an *oracle* that, given a line segment, detects whether the segment intersects the surface and, in the affirmative, returns an intersection point. This makes the algorithm useful in a wide variety of contexts and for a large class of surfaces.

The first contribution of our paper is to revisit the problem of computing the static visual hull by using the above meshing algorithm. To do so, we have designed a surface oracle and a refinement criterion adapted to multi-view geometry. The resulting algorithm both relates to volume-based and surface-based approaches. Similarly to the volume-based approach, our method builds a decomposition of space, namely the Delaunay triangulation of a point sample of the visual hull. Yet, it is an adaptive unstructured tetrahedral decomposition, in contrast with the usual voxel or octree decomposition, thus eliminating quantization artefacts. Similarly to the surface-based approach, our method directly outputs a surface mesh representation, in our case a high-quality watertight triangular mesh. Yet, it is a discrete approximation of the visual hull, under a controlled reprojection error, in contrast with an exact polyhedral intersection, thus avoiding numerical instability and degeneracies in the case of noisy or mutually inconsistent silhouette data. Moreover, compared to existing static visual hull techniques, our approach has the advantage of offering easy and extensive control over the size and quality of the output mesh as well as over its associated reprojection error.

The second and main contribution of our paper is a method to compute a 4D representation of the spatio-temporal visual hull of a non-rigid dynamic scene, based on an extension of the meshing algorithm of Boissonnat and Oudot [5, 6] to 4D. By considering time as an additional dimension, our approach exploits seamlessly the time coherence between different frames to produce a compact and high-quality 4D mesh representation of the visual hull. The 3D visual hull at a given time instant is easily obtained by intersecting this 4D mesh with a temporal plane.

Compared to independent frame-by-frame computations, our method has several significant advantages. *First,* it exploits time redundancy to limit the size of the output representation. For example, parts of the scene that are immobile or have a uniform motion can be approximated by a piecewise-linear 4D mesh with few elements elongated in the time direction. We will show in the following that this idea is related to non-uniform meshing depending on the spatio-temporal curvature. In contrast, in the same configuration, a frame-by-frame approach would repeat 3D elements at each frame. *Second,* our method yields a temporally continuous representation, which is defined at any time, thus enabling interpolation of objects' shape between consecutive frames. This also makes a spatio-temporal smoothing of visual hull possible, in order to recover from occasional outliers in silhouette data. *Third,* a byproduct of the two first advantages is the reduction of flicking artefacts in synthesized views, as consecutive 3D slices have a similar geometry and connectivity by construction.

A third contribution of our work is to demonstrate the feasibility of 4D hypersurface representations in computer vision. It is likely to inspire progress in other applications, such as spatio-temporal multi-view stereovision or segmentation of 3D+time MRI sequences of the heart in medical imaging. In that sense, the work of Goldlücke and Magnor [15] on spatio-temporal multi-view stereovision is related to ours. These authors take advantage of the fact that the level set method [25] easily extends to any number of dimensions. However, their method comes at a very high computation and memory cost, typically requiring several hours if not a day of computation on a cluster to process a few seconds of video. In contrast, despite the fact that our prototype implementation could be heavily optimized, our algorithm builds a compact 4D mesh representation of a few minutes of video in a much shorter computation time on a recent workstation.

The remainder of this paper is organized as follows. Section 2 gives some background on the basic computational geometry concepts needed in our approach: Voronoi diagrams, Delaunay triangulations and restricted Delaunay triangulations. Our novel methods for static and dynamic visual hull are described in Section 3. In Section 4, we report on some numerical experiments which demonstrate the effectiveness and flexibility of our approach for generating compact, high-quality, time-coherent visual hull representations from real silhouette image data.

2. Background

2.1. Voronoi Diagram and Delaunay Triangulation

Most of the following definitions are taken from [5]. We also refer the interested reader to some computational geometry textbooks [7, 11]. In the sequel, we call *k-simplex* the convex hull of $k+1$ affinely independent points. For example, a 0-simplex is a point, a 1-simplex is a line segment, a 2-simplex is a triangle and a 3-simplex is a tetrahedron. In this paper, we will also consider 4-simplices: they are known as *pentachorons* or *pentatopes*.

Let $E = \{p_1, \ldots, p_n\}$ be set of points in $\mathbb{R}^d$. Note that in this work, we are mainly interested in $d = 3$ and $d = 4$. The *Voronoi region*, or *Voronoi cell*, denoted by $V(p_i)$, associated to a point p_i is the region of space that is closer from p_i than from all other points in E:

$$V(p_i) = \{p \in \mathbb{R}^d \ : \ \forall j, \ \|p - p_i\| \leq \|p - p_j\|\} \ . \qquad (1)$$

$V(p_i)$ is the intersection of $n-1$ half-spaces bounded by the bisector planes of segments $[p_i p_j], \ j \neq i$. $V(p_i)$ is therefore a convex polytope, possibly unbounded. The *Voronoi diagram* of E, denoted by $\text{Vor}(E)$, is the partition of space induced by the Voronoi cells $V(p_i)$.

See Figure 1*(a)* for a two-dimensional example of a Voronoi diagram. In two dimensions, the edges shared by two Voronoi cells are called *Voronoi edges* and the points shared by three Voronoi cells are called *Voronoi vertices*. Similarly, in three dimensions, we term *Voronoi facets, edges* and *vertices* the geometric objects shared by one, two and three Voronoi cells, respectively. The Voronoi diagram is the collection of all these k-dimensional objects, with $0 \leq k \leq d$, which we call *Voronoi objects*. In particular, note that Voronoi cells $V(p_i)$ correspond to d-dimensional Voronoi objects.

The *Delaunay triangulation* $\text{Del}(E)$ of E is defined as the geometric dual of the Voronoi diagram: there is an edge between two points p_i and p_j in the Delaunay triangulation if and only if their Voronoi cells $V(p_i)$ and $V(p_j)$ have a non-empty intersection. It yields a *triangulation* of E, that is to say a partition of the convex hull of E into d-dimensional simplices (i.e. into triangles in 2D, into tetrahedra in 3D, into pentatopes in 4D and so on).

The fundamental property of the Delaunay triangulation is called the *empty circle* (resp. *empty sphere* in 3D, resp. *empty hypersphere* in higher dimensions) property: in 2D (resp. in 3D, resp. in 4D), a triangle (resp. tetrahedron, resp. pentatope) belongs to the Delaunay triangulation if and only if its circumcircle (resp. circumsphere, resp. circumscribed hypersphere) does not contain any other points of E in its interior.

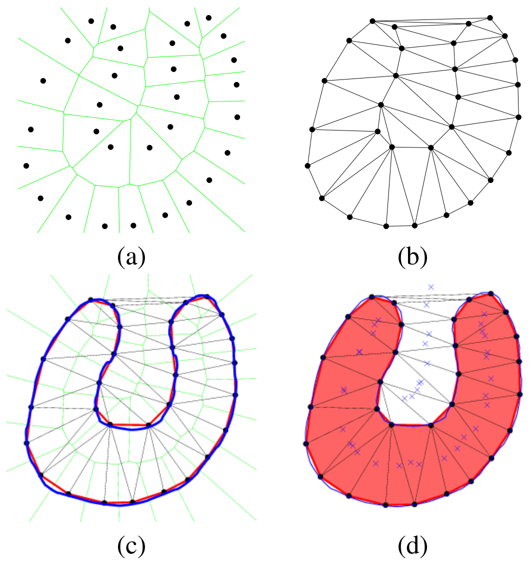

Figure 1. *(a)* Voronoi diagram of a set of points in the plane. *(b)* Its dual Delaunay triangulation. *(c)* The Delaunay triangulation restricted to the blue curve is plotted with a thick red line. *(d)* The Delaunay triangulation restricted to the region bounded by the blue curve is composed of the filled red triangles, whose circumcenters (blue crosses) are inside the region.

The algorithmic complexity of the Delaunay triangulation of n points is $\mathcal{O}(n \log n)$ in 2D, and $\mathcal{O}(n^2)$ in 3D. In 4D, there exist algorithms with $\mathcal{O}(n^3)$ complexity. As was recently proven in [2], the complexity in 3D drops to $\mathcal{O}(n \log n)$ when the points are distributed on a smooth surface, which is the case of interest here. A similar complexity drop can be expected in the 4D case, although we are not aware of theoretical results in this direction.

2.2. Restricted Delaunay triangulation

Each k-simplex in the Delaunay triangulation is dual to a $(d-k)$-dimensional Voronoi object. In 3D, the dual of a Delaunay tetrahedron is the Voronoi vertex which coincides with the circumcenter of the tetrahedron, the dual of a Delaunay facet is a Voronoi edge, the dual of a Delaunay edge is a Voronoi facet, and the dual of a Delaunay vertex p_i is the Voronoi cell $V(p_i)$.

Given a subset $\Omega \in \mathbb{R}^d$, typically a manifold of dimension $k \leq d$, we call the *Delaunay triangulation of E restricted to* Ω, and we note $\text{Del}|_\Omega(E)$ the subcomplex of $\text{Del}(E)$ composed of the Delaunay simplices whose dual Voronoi objects intersect Ω. For example, in 2D, as illustrated in Figure 1*(c)*, the Delaunay triangulation restricted to a curve C is composed of the Delaunay edges whose dual Voronoi edges intersect C. Similarly, as shown in Figure 1*(d)*, the Delaunay triangulation restricted to a region R is composed of the Delaunay triangles whose circumcenters are contained in R. The attentive reader may have noticed that in both cases the restricted Delaunay triangulation forms a good approximation of the object.

Actually, this is a general property of the restricted Delaunay triangulation. It can be shown that, under some assumptions, and especially if E is a "sufficiently dense" sample of Ω, in some sense defined in [1], $\text{Del}|_{\Omega}(E)$ is a good approximation of Ω, both in a topological and in a geometric sense: as regards topology, $\text{Del}|_{\Omega}(E)$ is homeomorphic to Ω; as regards geometry, the Hausdorff distance between $\text{Del}|_{\Omega}(E)$ and Ω can be made arbitrarily small; normals and curvatures of Ω can be consistently approximated from $\text{Del}|_{\Omega}(E)$.

Based on these approximation properties, a family of provably correct algorithms for mesh generation and mesh reconstruction from point clouds have been designed in the last decade. We refer the reader to [5] and references therein for more details.

3. Methods

In this section, we first present our novel method for computing the static visual hull, based on the incremental 3D meshing algorithm of Boissonnat and Oudot [5, 6]. Then, we describe in detail the extension of this method to the computation a 4D representation of the spatio-temporal visual hull of a non-rigid dynamic scene.

3.1. Static Visual Hull Computation

Let $I_i,\ i \in \{1, \ldots, n\}$ be the set of input images, and let $\Pi_i : \mathbb{R}^3 \to \mathbb{R}^2$ denote the camera projections from the world reference frame to the image planes. In addition, $\Omega_i \subset \mathbb{R}^2$ are the silhouettes of the object in the different images. The visual hull V is then defined by:

$$V = \left\{p \in \mathbb{R}^3 \mid \forall i \in \{1, \ldots, n\},\ \Pi_i(p) \in \Omega_i\right\} . \tag{2}$$

Let us consider a set E of points lying on the boundary of the visual hull ∂V. Our approach consists in approximating the visual hull by the Delaunay triangulation of E restricted to V, *i.e.* in computing the union of tetrahedra of $\text{Del}(E)$ whose circumcenters are contained in the visual hull. The output triangular mesh is then obtained by considering the boundary facets of this set of tetrahedra. Interestingly, this directly enforces watertight surface meshes free of self-intersections. As mentioned above, it can be proven that this mesh forms a good approximation of the visual hull as soon as E is a "sufficiently dense" sample of ∂V. With this procedure in hand, our visual hull algorithm reduces to generating a point sample E which fulfills the above sampling condition as well as some additional user-defined quality and error criteria on boundary facets.

Our algorithm for generating E closely parallels the surface meshing algorithm of Boissonnat and Oudot [5, 6]. The algorithm starts with a small initial point sample E_0 of ∂V (different strategies to generate the latter are detailed in [5, 6]) and, at each iteration, it inserts a new point of ∂V into E and updates $\text{Del}|_V(E)$. Each point inserted into E is the intersection between ∂V and the dual of a boundary facet (that is to say, a ray or a segment of the Voronoi diagram of E). Note that such an intersection always exists, by construction. In case there are several intersections, any of them can be chosen, without compromising the good continuation of the algorithm. The algorithm stops when there are no bad boundary facets left.

The initial computation of $\text{Del}|_V(E)$ and its subsequent updates are fairly simple: it suffices to project the circumcenters of Delaunay tetrahedra in all images and to check whether all the projections lie inside the silhouettes. This check is only performed for tetrahedra that have been modified by the previous point insertion. Similarly, intersections of a segment or a ray with the boundary of the visual hull are computed to the desired accuracy using a dichotomic search along their projection in the different input images.

Under these considerations, the overview of our algorithm is given below:

```
while there is a bad boundary facet do
    let $f$ be the worst boundary facet
    let $p$ be an intersection between $\partial V$ and the dual of $f$
    insert $p$ in $E$
    update $\text{Del}|_V(E)$
end while
```

In the above algorithm, the determination of "good" and "bad" boundary facets is devoted to some user-defined criteria, for example a combination of thresholds on the following elementary quality measures: aspect ratio (minimum angle), size, curvature, edge length, etc. But the most relevant quality criterion in our case is undoubtedly the reprojection error, that is to say the discrepancy between the input silhouettes and the silhouettes of the computed mesh.

To this end, we use the signed distance functions $\phi_i : \mathbb{R}^2 \to \mathbb{R}$ associated to the input silhouettes Ω_i. In other words, for a point x in image i, we set:

$$\begin{cases} \phi_i(x) = -d(x, \partial\Omega_i) & \text{if} \quad x \in \Omega_i \\ \phi_i(x) = +d(x, \partial\Omega_i) & \text{if} \quad x \notin \Omega_i \end{cases} , \tag{3}$$

with $d(x, \partial\Omega_i)$ the distance from the point $x \in I_i$ to the boundary of the silhouette Ω_i. Let us note $\Phi : \mathbb{R}^3 \to \mathbb{R}$ the maximum of the reprojected distance functions, *i.e.* $\Phi(x) = \max_i\ \phi_i \circ \Pi_i(x)$. An alternate definition of the visual hull can be written in terms of the Φ function:

$$V = \left\{x \in \mathbb{R}^3 \mid \Phi(x) \leq 0\right\} . \tag{4}$$

Interestingly, the maximum reprojection error of a boundary facet with respect to input silhouettes can also be measured with Φ:

$$\text{error}(f) = \max_{x \in f} |\Phi(x)| . \tag{5}$$

In practice, this error measure is computed by sampling triangular facets and collecting the maximum value of the different ϕ_i at reprojected locations.

During the progress of the algorithm, boundary facets whose error measure exceeds a user-defined threshold (typically one pixel) are further refined. As a result, when the algorithm terminates, the whole output mesh fulfills the desired bound on reprojection error.

3.2. Spatio-Temporal Visual Hull Computation

Given multiple video sequences of a moving object and its silhouettes extracted in all images, we could run the above algorithm (or any other static visual hull algorithm) independently in each time frame, and obtain a sequence of 3D visual hull meshes. As discussed previously, this frame-by-frame approach has some significant deficiencies, because it does not exploit temporal coherence. Here, we propose to construct a "global" spatio-temporal mesh of the sequence, consistent with all input silhouettes.

The main idea of our algorithm is to regard time as a fourth dimension, and to treat it similarly to the three spatial dimensions. At first sight, this is questionable since space is not homogeneous to time regarding physical units. We obtain physical homogeneity of our 4D space by considering a scaling factor v_0 between space and time dimensions. This scaling factor is homogeneous to a speed, and can be interpreted as a reference displacement per time unit, beyond which objects' temporal behavior is not regarded as a continuous motion. So there is no difficulty to tune this parameter for human motion.

We now extend all the definitions of Section 3.1 to the dynamic case. Let $I_i^t,\ i \in \{1,\dots,n\},\ t \in [0,T]$ denote the input video sequences, and $\Omega_i^t \subset \mathbb{R}^2$ the corresponding silhouettes. The spatio-temporal visual hull V is then defined by:

$$V = \left\{(p, v_0 t) \in \mathbb{R}^3 \times [0, v_0 T] \mid \forall i,\ \Pi_i(p) \in \Omega_i^t\right\} \ . \quad (6)$$

There is an important remark we can make here. Actually I_i^t and Ω_i^t are only known at discrete time instants $k\Delta t$, the video frame rate being $1/\Delta t$. A similar remark holds for image locations, but it is likely to be of less practical importance given the increasing resolution of standard consumer video cameras.

Our method for computing an approximating 4D mesh of the spatio-temporal visual hull is a careful extension of the static case. We incrementally build a "good" 4D point sample E of the boundary of the spatio-temporal visual hull ∂V and we maintain the 4D Delaunay triangulation of E restricted to V, in other words the union of the *pentatopes* of Del(E) whose circumcenters lie inside V. It suffices to project the circumcenters of lastly modified pentatopes in the different cameras, and to check whether the projections belong to the silhouettes.

As the fourth coordinate of these circumcenters is unlikely to correspond to an available time frame, an interpolation of silhouettes between consecutive frames is required. To this end, we use a linear interpolation of the signed distance functions $\phi_i^{k\Delta t}$, which is an established technique in shape statistics [21, 27].

The output 4D mesh is then obtained by considering the boundary facets of the pentatopes of Del$|_V(E)$. The exact nature of these "facets" deserves clarification: they are tetrahedra with 4D coordinates, so they are indeed *simplicial* pieces of a hypersurface in $\mathbb{R}^4$. As in the static visual hull algorithm, the construction of E consists in iteratively adding an intersection point between ∂V and the 4D segment/ray dual to the worst boundary tetrahedron. In practice, a dichotomic search is used along with the aforementioned time interpolation of silhouettes. Again, the refinement process is controlled by a threshold on the reprojection error of boundary facets. The algorithm terminates as soon as the obtained 4D mesh fulfills the desired reprojection error with respect to all silhouettes of the sequence.

At this point, there are several important remarks to be made. First, the attentive reader may have noticed that the vertices of the 4D output mesh generally do not lie in the input temporal planes $t = k\Delta t$. Second, the space and time density of these spatio-temporal vertices is fully adaptive. Typically, it will be coarse in parts of the visual hull of low curvature and/or of uniform motion. The underlying idea is that regions of ∂V of low *spatio-temporal curvature* can be well approximated with few boundary facets. This allows to keep the total size of the 4D representation of the scene, and hence the computational and memory cost, sustainable.

Our third remark is intended to overcome doubts about the relevance of interpolating silhouettes between consecutive time frames. These doubts are grounded in the case of a very low frame rate relatively to scene motion. This said, our approach is intended for highest possible frame rates: indeed, our method has the remarkable property that the increase in the number of input frames does not affect either the computational expense or the size of the output. Actually, only complexity of scene geometry and motion matters.

The output 4D mesh cannot be used directly for rendering. Fortunately, the 3D visual hull at a given time instant is easily obtained by intersecting this 4D mesh with a temporal plane. This task can be performed very efficiently, even in real-time on GPUs (Graphics Processor Units), since it reduces to a *marching tetrahedra* algorithm [17] on the tetrahedra of the 4D mesh, with the temporal coordinate of vertices used as the scalar field for isocontouring. It produces one triangle or one quad per boundary tetrahedron intersected by the selected temporal plane. Note that the produced 3D vertices do not coincide with vertices of the 4D mesh, they are linear interpolations of the latter. Also, their

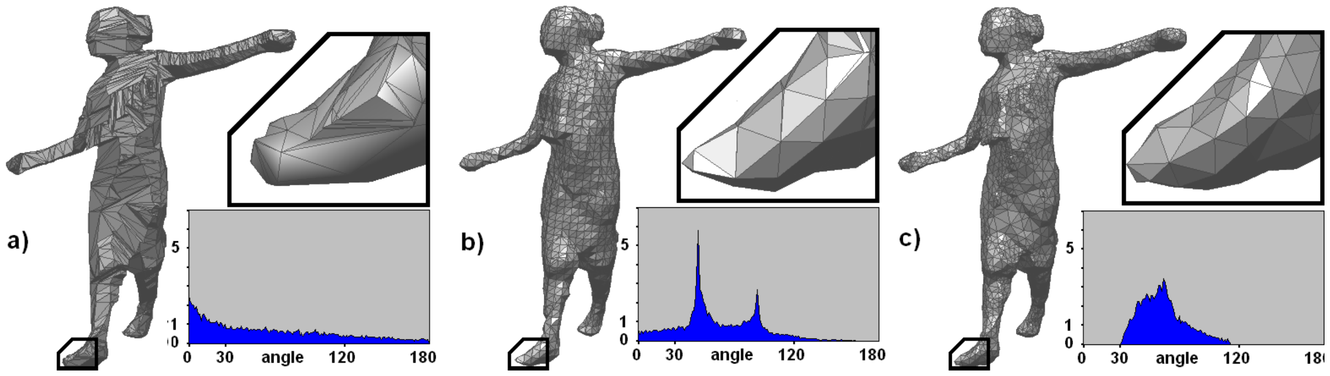

Figure 2. Static Visual Hull Reconstruction produced by a) the method introduced in [12], b) a volume-based approach followed by marching cube algorithm, c) our method. The magnified views exemplify some typical mesh configurations produced by the three approaches: a) a large number of low quality triangles; b) a uniform resolution mesh containing numerous skinny triangles; c) a non-uniform high-quality mesh. The histograms represent the angle distributions of the obtained surface meshes; the quality of the visual hull meshes is much higher with our approach than with other algorithms.

position and connectivity vary continuously with time.

3.3. Implementation Aspects

We have implemented our approach using *CGAL* (Computational Geometry Algorithms Library, homepage: www.cgal.org) [4]. CGAL defines all the needed geometric primitives and provides an excellent algorithm to compute the Delaunay triangulation in 3D: it is robust to degenerate configurations and floating-point error, thanks to the use of exact geometric predicates, while being able to process millions of points per minute on a standard workstation. CGAL also provides a generic unoptimized Delaunay triangulation algorithm which works in any number of dimensions. We have used this code to implement our spatio-temporal visual hull method. When used in 3D, this generic code runs 20 to 30 times slower than CGAL's dedicated 3D Delaunay code. Consequently, a two orders of magnitude reduction in computation time can be expected in the future, after developing an optimized 4D Delaunay code. Surprisingly, despite the frequent occurrence of spatio-temporal data in science and engineering, no such code is publicly available.

As for the signed distance functions of input silhouettes, we compute them efficiently using *fast marching* [28].

4. Experimental Results

We have tested our algorithms on some real datasets, publicly available at `https://charibdis.inrialpes.fr/`.

4.1. Static 3D Visual Hull Reconstruction

Our first experiment focuses on static visual hull reconstruction. We have used the first frame of the "Danse" sequence recorded by eight cameras with resolution 780x582 pixels. In order to illustrate the effectiveness and flexibility of our approach, we compare it to two popular methods, a surface-based approach by Franco and Boyer based on exact polyhedral intersection [12], and a volume-based approach followed by a marching cubes algorithm. To provide a fair comparison between the three methods, we adapted the resolution of the 3D image used for the volume-based approach and the refinement criteria of our method (the maximum reprojection error was chosen to be $\max_{x\in f} |\Phi(x)| \leq 0.98$) so that all three meshes would have approximately the same number of vertices. The quantitative results (number of vertices, number of faces, and computation time) are gathered in Table 1. Figure 2 displays the reconstructed visual hulls.

Experiment	# points	# triangles	Time (in sec.)
[12]	2400	4790	0.14
Marching cubes	2400	4840	0.04
Our method	2400	4770	2.0

Table 1. Parameters and quantitative results of our different numerical experiments.

In order to show the quality of the output surface meshes, we computed their angle distribution. Figure 2 shows the three histograms. Contrarily to our method, which produces well-shaped triangles only, other methods yield meshes with lots of skinny triangles.

4.2. Spatio-Temporal Visual Hull Computation

In a second experiment, we tested our spatio-temporal visual hull algorithm on a real human motion dataset. we have choosen the "Nicolas2" sequence from the *INRIA Xmas Motion Acquisition Sequences* (IXMAS) dataset. The sequence contains 1084 frames recorded by 5 standard Firewire cameras with resolution 390x291 pixels. We use the silhouette data included in the dataset, which is signif-

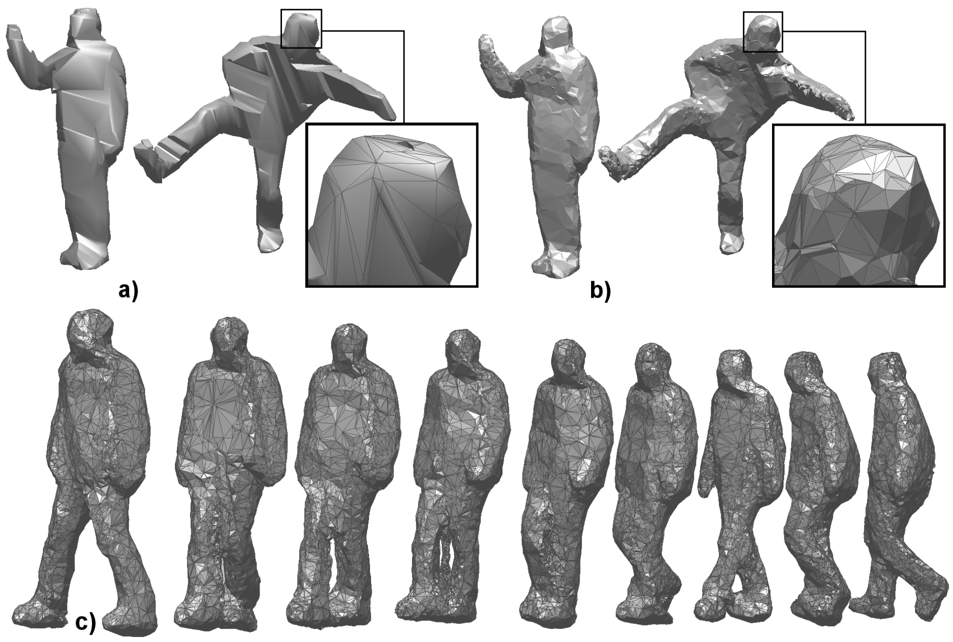

Figure 3. Some sample frames of the temporal visual hull reconstruction produced by: a) the method of Franco and Boyer [12], b) and c) our method. The magnified views exemplify the significantly higher accuracy offered by our approach, with an equivalent number of vertices.

icantly corrupted by noise and errors, in order to test the robustness of our approach. Figure 3 displays a comparison between our method and the method of Franco and Boyer [12], applied independly in each frame. We compared the 3D meshes output by their method for some frames of the sequence with the corresponding 3D slices of our 4D spatio-temporal visual hull mesh. To provide a fair comparaison, we adapted the spatio-temporal scaling factor v_0 and the refinement criteria of our method (the maximum reprojection error was chosen to be $\max_{x\in f} |\Phi(x)| \leq 1.00$ with a scaling factor $v_0 = 0.72$ (space unit/s)) so that the two methods would construct the whole sequence with approximately the same number of vertices (the number of vertices of our 4D mesh is compared to the total number of vertices of the 3D meshes for the different frames). Table 2 shows the quantitative results of these experiments.

It appears that, despite an equivalent number of vertices in the ouput representation, our method yields a much more accurate visual hull. This is due to the fact that it generates a much higher number of "apparent vertices" in the 3D time slices than the actual number of 4D vertices. This perfectly illustrates the capability of our approach to take advantage of temporal coherence in order to generate a more compact spatio-temporal representation. Finally, let us mention that our method naturally handles topology changes of the visual hull along time, as visible in Figure 3 c).

Experiment	# points	# points per frame (mean)	Time (minutes)
[12]	769356	709	3
Our method	848376	4540 (apparent)	209

Table 2. Parameters and quantitative results of our temporal visual hull experiment.

5. Discussion and Conclusion

We have proposed a method to compute the spatio-temporal visual hull of a non-rigid dynamic scene, based on four-dimensional Delaunay meshing. Our method outputs a compact and temporally coherent 4D mesh representation of the whole scene. 3D slices of this mesh can easily be computed for rendering and for interpolating shape between consecutive time frames.

We have validated our approach on real video sequences of human motion. Our results compare favorably with state of the art methods in terms of compactness and quality of the mesh representation. Our work demonstrates the feasibility of 4D hypersurface representations in computer vision. To that extent, we believe that it can inspire progress in other spatio-temporal problems, such as multi-view stereo-vision of dynamic scenes, or segmentation of 3D+time MRI sequences of the heart in medical imaging.

A significant drawback of our method is that it is not adapted to the online processing of video sequences. The first reason is its high computational expense. This said, in the future, we expect a two orders of magnitude reduction in computation time thanks to an optimized 4D Delaunay code, thus making interactive processing possible. The second reason is that the very principle of our approach is to treat the video sequence as a whole, thus making offline processing mandatory. Obviously, this approach is not sustainable when the length of the video sequence increases. To overcome this limitation, we consider taking inspiration in the work of Isenburg *et al.* on streaming computation of Delaunay triangulations [18]. The principle of such an approach would be to restrict computations to a short sliding time window, in order to produce streaming 4D mesh output from streaming video input.

References

[1] N. Amenta and M. Bern. Surface reconstruction by Voronoi filtering. *Discrete and Computational Geometry*, 22:481–504, 1999.

[2] D. Attali, J.-D. Boissonnat, and A. Lieutier. Complexity of the Delaunay triangulation of points on surfaces: the smooth case. In *Annual Symposium on Computational Geometry*, pages 201–210, 2003.

[3] B. Baumgart. *Geometric Modeling for Computer Vision*. PhD thesis, Stanford University, 1974.

[4] J.-D. Boissonnat, O. Devillers, M. Teillaud, and M. Yvinec. Triangulations in CGAL. In *Annual Symposium on Computational Geometry*, pages 11–18, 2000.

[5] J.-D. Boissonnat and S. Oudot. Provably good sampling and meshing of surfaces. *Graphical Models*, 67:405–451, 2005.

[6] J.-D. Boissonnat and S. Oudot. Provably good sampling and meshing of Lipschitz surfaces. In *Annual Symposium on Computational Geometry*, 2006.

[7] J.-D. Boissonnat and M. Yvinec. *Algorithmic Geometry*. Cambridge University Press, 1998.

[8] E. Boyer and J.-S. Franco. A hybrid approach for computing visual hulls of complex objects. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 695–701, 2003.

[9] K. Cheung, S. Baker, and T. Kanade. Shape-from-silhouette across time: Part I: Theory and algorithms. *The International Journal of Computer Vision*, 62(3):221–247, 2005.

[10] K. Cheung, S. Baker, and T. Kanade. Shape-from-silhouette across time: Part II: Applications to human modeling and markerless motion tracking. *The International Journal of Computer Vision*, 63(3):225–245, 2005.

[11] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry, Algorithms and Applications*. SpringerVerlag, 1997.

[12] J.-S. Franco and E. Boyer. Exact polyhedral visual hulls. In *British Machine Vision Conference*, volume 1, pages 329–338, 2003.

[13] J.-S. Franco and E. Boyer. Fusion of multi-view silhouette cues using a space occupancy grid. In *International Conference on Computer Vision*, volume 2, pages 1747–1753, 2005.

[14] J.-S. Franco, M. Lapierre, and E. Boyer. Visual shapes of silhouette sets. In *3D Processing, Visualization and Transmission*, 2006.

[15] B. Goldlücke and M. Magnor. Space-time isosurface evolution for temporally coherent 3D reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 350–355, 2004.

[16] L. Guan, S. Sinha, J.-S. Franco, and M. Pollefeys. Visual hull construction in the presence of partial occlusion. In *3D Processing, Visualization and Transmission*, 2006.

[17] A. Guéziec and R. Hummel. Exploiting triangulated surface extraction using tetrahedral decomposition. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):328–342, 1995.

[18] M. Isenburg, Y. Liu, J. Shewchuk, and J. Snoeyink. Streaming computation of Delaunay triangulations. In *ACM SIGGRAPH*, pages 1049 – 1056, 2006.

[19] A. Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(2):150–162, 1994.

[20] S. Lazebnik, E. Boyer, and J. Ponce. On computing exact visual hulls of solids bounded by smooth surfaces. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 156–161, 2001.

[21] M. Leventon, E. Grimson, and O. Faugeras. Statistical shape influence in geodesic active contours. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 316–322, 2000.

[22] W. Lorensen and H. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM Computer Graphics*, 21(4):163–170, 1987.

[23] W. Matusik, C. Buehler, and L. McMillan. Polyhedral visual hulls for real-time rendering. In *Eurographics Workshop on Rendering*, pages 115 – 126, 2001.

[24] W. Matusik, C. Buehler, R. Raskar, S. Gortler, and L. McMillan. Image based visual hulls. In *ACM SIGGRAPH*, pages 369–374, 2000.

[25] S. Osher and J. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton–Jacobi formulations. *Journal of Computational Physics*, 79(1):12–49, 1988.

[26] M. Potmesil. Generating octree models of 3D objects from their silhouettes in a sequence of images. *Computer Vision, Graphics and Image Processing*, 40(1):1–29, 1987.

[27] M. Rousson and N. Paragios. Shape priors for level set representations. In *European Conference on Computer Vision*, volume 2, pages 78–92, 2002.

[28] J. Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93(4):1591–1694, 1996.

[29] D. Snow, P. Viola, and R. Zabih. Exact voxel occupancy with graph cuts. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 345–352, 2000.

[30] R. Szeliski. Rapid octree construction from image sequences. *Computer Vision, Graphics and Image Processing*, 58(1):23–32, 1993.

[31] B. Vijayakumar, D. Kriegman, and J. Ponce. Structure and motion of curved 3d objects from monocular silhouettes. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 327–334, 1996.