A Novel Junction-aware Shape Descriptor for 3D Articulated Models with Visible Internal Volume

Yu-Shen Liu, Hong-Chen Deng, Min Liu, Lian-Jie Gong, Jean-Claude Paul

Abstract—An articulated shape is composed of a set of rigid parts connected by some flexible junctions. The junction has been demonstrated to be a critical local feature in many visual tasks such as feature recognition, segmentation, matching, motion tracking and functional prediction. However, efficient description and detection of junctions still remain a research challenge due to high complexity of articulated deformation. This paper presents a novel junction-aware shape descriptor for a 3D articulated model defined by a closed manifold surface. To encode junction information on the shape boundary, the core idea is to develop a new geometric measure, called the visible internal volume (VIV) function, which associates the shape's volumetric context to its boundary surface. The VIV at an arbitrary point on the shape boundary is defined as the volume of visible region within the shape as observed from the point. The VIV variation serves as the new shape descriptor. One advantage of using the VIV for 3D articulated shape description is that it is robust to articulation and it reflects the shape structure and deformation well without any explicit shape decomposition or prior skeleton extraction procedure. The experimental results and several potential applications are presented for demonstrating the effectiveness of our method.

Index Terms—3D articulated shapes, junction detection, shape descriptor, visible internal volume (VIV), visibility graph.

1 Introduction

Non-rigid shapes are ubiquitous in the world and, due to their physical properties, can undergo variant deformation [1]. Non-rigid shape analysis has been receiving a growing amount of attention in many applications in computer vision, computer graphics, pattern recognition and molecular structure analysis [1], [2]. A simplified approach to non-rigid shape analysis is based on *articulated shapes* [2]–[6], which assumes that the non-rigid shape is composed of a set of rigid parts connected by some flexible junctions (or named *joints/hinges* in some literature [2], [7]). Each of rigid parts has a certain degree of freedom to move, and junctions are relatively small compared to parts they connect.

The junction, as a critical local feature, provides valuable local information for analyzing 3D articulated shapes. Many applications, such as feature recognition, segmentation, matching, motion tracking, functional prediction and folding detection, can benefit from automatic detection of junctions. For instance, if the appropriate junctions and parts can be detected, they can help to improve the performance of articulated shape segmentation [8], [9]. In medical applications, many visual tasks such as vessel tracking may rely on junction extraction, which can facilitate diagnosis and understanding of pulmonary vascular diseases [10], [11]. In molecular structure analysis, many flexible molecules can be viewed as 3D articulated objects consisting of some rigid parts connected by rotary hinges/junctions [7], [12], [13], where these rigid parts may swivel with respect to each other around the hinge sites. Identification of hinge and binding sites is a fundamental problem at protein functional prediction and comparison [7], [12]–[15]. The abilities to infer junction information of 3D articulated shapes have proven to be useful in different application areas. However, efficient description and detection of junctions still remain a research challenge due to high complexity of articulated deformation.

There are several possible ways to solve this issue. One way is, generally in protein structure analysis, to analyze one or more protein structures, frequently with the help of additional chemical information. However, such structures are not explicitly available for a general 3D articulated model in computer vision and computer graphics. An alternative way is to first extract the topological skeleton of a 3D object and then identify additionally its junctions and branches based on skeleton analysis. Nevertheless, computing a robust skeleton is non-trivial. The third way is based on shape segmentation methods, whereas identifying a clear-cut junction is not an easy task. Another attractive way is based on shape descriptor methods, in which most prior work has focused on detecting local surface features and interest regions alone. Although a few of recent articles attempted to consider the volume-oriented shape descriptors with respect to articulated deformation, they are not junction-aware yet. In general, no shape descriptor employed so far encodes

- *Y.S. Liu is with the School of Software, Tsinghua University, Beijing 100084, China; Key Laboratory for Information System Security, Ministry of Education; Tsinghua National Laboratory for Information Science and Technology. E-mail: liuyushen@tsinghua.edu.cn. Homepage: http://cgcad.thss.tsinghua.edu.cn/liuyushen/.*
- *H.C. Deng and L.J. Gong are with the School of Software, Tsinghua University. E-mail: denghc09@gmail.com, lianjiegong@gmail.com.*
- *M. Liu is with the Institute of Manufacturing Engineering, Tsinghua University. E-mail: minliu@tsinghua.edu.cn.*
- *J.C. Paul is with the School of Software, Tsinghua University; INRIA, France. E-mail: paul@tsinghua.edu.cn.*

junction information on 3D articulated shapes explicitly.

This paper presents a novel junction-aware shape descriptor for a 3D articulated object defined by a closed manifold surface. To encode junction information on the shape boundary, the core idea is to develop a new geometric measure, called the visible internal volume (VIV) function, which considers the volumetric context inside the shape. The VIV at an arbitrary point on the shape boundary is defined as the volume of visible region as observed from the point. The denoised and filtered variation of VIV serves as the base of our new shape descriptor. One advantage of using the VIV function for 3D articulated shape description is that it is robust to articulation and it reflects the shape structure and deformation well without any explicit shape decomposition or prior skeleton extraction procedure. We have tested our method and compared it with some relevant approaches using several well-known 3D articulated shape databases, such as Princeton Segmentation Benchmark (PSB) [16] and ISDB [17]. The experimental results and several applications are presented for demonstrating the effectiveness of our method.

Note that a class of junction detection methods is based on shape alignment/comparison techniques, which first compare two or more different shapes and then determine the correspondences and junctions. Unlike the comparison-based techniques, our method only resorts to the geometric representation of the given individual shape. For this reason, the proposed method can be computed off-line and this property is of great importance to many further applications that require fast and effective processing.

2 Related work

Although the main issue discussed in the paper focuses on junction description for 3D objects, a large number of previous techniques are connected to 2D images. Therefore, reviewing the 2D case will provide a good understanding for the 3D situation.

2.1 Junction detection in 2D images

In 2D images, junctions can be classified by the number of wedges [18], including two-junctions (e.g. corners), three-junctions (e.g. T or Y-junctions), four-junctions (e.g. X-junctions), and so on. These junctions are the important features for image analysis and form a critical aspect of image understanding tasks such as object recognition [18], [19]. *Junction detectors/descriptors* do the job of describing and identifying both locations and information of junctions [19]. A well-known junction detector is *corner detector*. In general, there are two basic approaches for detecting junctions: *region-based* approaches and *edge-based* ones.

In the region-based approaches, a junction is usually modelled as a region of an image where some wedge-shaped regions meet [18]. For instance, Parida et al. [18] presented a unified approach for detecting, classifying and reconstructing junctions in images. Their approach used a template deformation framework for junction description. Yu et al. [20] developed a technique to characterize junctions based on applying rotated copies of a wedge averaging filter and estimating the derivative with respect to the polar angle. Ruzon and Tomasi [21] described an algorithm for junction detection using the color distribution in color images. Recently, Su et al. [10] proposed a method for detecting junctions in 2D images with linear structures such as blood vessels and plant roots, where junctions of linear structures are where three or more branches intersect.

In the edge-based approaches, a junction is alternatively described as the image points where two or more ridges meet [19], [22]. For instance, [23] described a method for junction detection in gray-level images by extracting the lines and estimating the local line curvature. The detector in [24] used a two-pass Hough transform to identify the intersection points in an edge map. More recently, Elias and Laganière [19], [22] proposed an edge-based junction detector named JUDOCA in 2D images.

Generally speaking, either the region-based or the edge-based approaches utilize characteristics which are usually available in 2D images for efficiently detecting junctions (e.g. the number of radial line boundaries, the angular direction of boundaries, the intensity within each wedge as well as the limited templates of junction deformation [18]). However, those characteristics are generally difficult to be obtained by only analyzing 3D geometric shapes. A survey of many available methods for junction detection in 2D images is beyond the scope of this paper. We only review some representative methods and refer the readers to the comprehensive introduction from Refs. [10], [18], [19], [22], [25].

2.2 Junction detection in 3D shapes

Although junction detectors/descriptors have been broadly studied in 2D images, only few papers explicitly discussed this issue for 3D articulated shapes. We will review some studies that are related to junction description and detection of 3D articulated shapes. In general, the existing approaches are dependent on 3D shape representation in specific applications. The related work can be summarized as follows.

Fitting-based methods. In medical applications, Zhao et al. [11] introduced a fitting-based technique for detecting the vessel junction of 3D CT data, where the parametric junction model is fitted using three tori patches and an ellipsoid surface. For 3D articulated shapes with complex structures, it is not an easy task to fit the junction region only using several simple quadratic surfaces.

Structure analysis methods. In protein structure analysis, there are two types of methods for detecting protein hinges/junctions. One way is to compare the structures of two or more proteins first, and then determine the corresponding junctions between them [7], [12], [13]. Another way is to predict the junction positions from

a single protein structure without comparing with other ones [14], [15]. However, junction prediction of a single protein is strongly dependent on the given protein representation, where a 3D protein structure is typically represented as an ordered set of its backbone C_α atoms (each atom has a 3D coordinate) or as its secondary structure. Meanwhile, many prediction algorithms often take advantage of some additional chemical information of protein structures, such as amino acid sequences. Our method is designed to treat the geometric representation of a 3D articulated shapes itself, where no prior structure information or any other non-geometric properties are assumed in our shape data.

Skeleton-based methods. In computer graphics, many techniques focus on topology or graph extraction of 3D shapes with surface representations (e.g. polygonal meshes [26]–[28]) or volumetric representations [28]. This extraction process produces a thin skeleton (or medial axis), possibly with additional junctions and branches corresponding to the logical components of the object [26], [27]. Nevertheless, the extracted skeleton is often very sensitive to perturbation and noise on the object's boundary. In addition, computing such a robust skeleton is not an easy task. Also, the skeleton typically induces only a rough junction position and the final junction region still needs to be computed on the boundary surface of the shape. Although several robust skeleton extraction methods may provide a prior reference for junction detection applications, it remains a separate research topic. In contrast, our method directly highlights the junction regions on the boundary surface without requiring any skeleton extraction procedure.

Segmentation-based methods. 3D shape segmentation/partition methods have gained much attention in recent years. Almost all of the methods on shape segmentation/partition have to face the same question of how to define parts or part boundaries of 3D objects [8], [29]–[31]. The classical part-type segmentation techniques analyze the geometric structure of an individual shape in order to detect parts or part boundaries (e.g. [8], [9], [30]), where a variety of geometric features have been investigated. Recently, another trend is to segment the shapes jointly, utilizing features from multiple similar shapes to improve the segmentation of each [29], [31]. In fact, there is also a strong connection between part-partitioning and skeletonizing [9], [31]. Many part-type segmentation methods are based on skeleton extraction and its structure analysis [9]. After dividing a 3D shape into meaningful parts, the intersection curve where two or more parts meet can assist to find the candidate junction. However, using part-type segmentation to assist junction identification is a type of "chicken-or-egg" dilemma, since the definition of a part implies the identification of clear-cut junctions. To solve this dilemma, our method produces a junction-aware shape descriptor without any explicit shape decomposition. Moreover, our descriptor, as an intrinsic measure, can also assist segmentation of 3D articulated shapes over pose changes.

Shape descriptor methods. Another alternative way to junction detection is based on shape descriptor. Our method falls into this category. A shape descriptor is a concise representation of the shape that expresses some of its specific properties [4], [17]. A simple realization of junction detection is based on measuring the local surface properties of the shape boundary, such as curvature, dihedral angles, or the local *shape-radius* (i.e. the radius of the maximal inscribed ball in the medial axis). However, these measures are limited by their local surface nature and are also very sensitive to noise. Recently, there have also been some efforts to develop the new shape descriptors for detecting the interesting and important surface regions or features, such as salient regions of surfaces [32], [33], distinctive regions of surfaces [34], [35], and local feature descriptors [36], [37]. However, they are still purely surface-based and do not take account of the volumetric information explicitly.

It is worthwhile to mention the *shape-diameter* function (SDF) [17], [30], [31] (or named *local-diameter* descriptor). The SDF is a scalar function defined on mesh surface, and it links the local volume to the boundary surface by approximating the shape diameter instead of computing the medial axis. The algorithm of computing the SDF can be conducted as follows. For each point $\mathbf{p}$ on the boundary surface, the algorithm shoots a cone of rays opposite to the normal of $\mathbf{p}$ and identifies their intersection points with the boundary surface; then the median or average length of all intersection rays is set as the shape diameter. One of our previous work has applied the SDF to flexible protein shape comparison [38]. As a shape descriptor, the SDF is pose-invariant but still not junction-aware, and we will explain this point later.

Generally speaking, most existing shape descriptors are not specifically designed to characterize junctions for 3D articulated models. Moreover, many of these descriptors typically relate to the measures of local surface convexities or concavities, but such convex and concave features on local surfaces do not completely reflect characteristics of junctions. A reasonable junction-ware descriptor should also capture the volumetric context of 3D shapes. An existing volume-oriented geometric metric is the inner distance [3], which measures the length of the shortest path between two surface points within the shape volume. By extending Ling and Jacobs's work [3] from 2D to 3D, our recent work has described a visibility graph-based algorithm for computing inner distances of 3D volumetric shapes [4], which also derived some applications [39]–[41]. Although the inner distance is articulation-invariant, it is not junction-aware. In contrast, this article develops a new volume-oriented measure (i.e. VIV) and quantifies its variation as a junction-aware shape descriptor. We will demonstrate that our new descriptor is junction-aware.

Another recent work related to ours is Ref. [8], in which a part-aware surface measure was developed by considering the volumetric context inside the shape. It

is based on an observation that the visible region as observed inside a shape's volume provide strong hints for part transition. The visible region generally remains stable within a part, while only changes greatly across part boundaries. Our notion is also inspired by such observation. The algorithm in [8] first computes the reference point $\mathbf{r}_i$ for each mesh facet by mapping the centroid of the facet onto the pre-calculated medial axis. Then the visible region as seen from each reference point $\mathbf{r}_i$ is approximated by a sampling approach, which consists of three steps: (1) m rays are uniformly sampled on a sphere surrounding $\mathbf{r}_i$; (2) m intersection points between the rays and the mesh boundary surface are collected in a set of $\{\mathbf{s}_i^{(1)}, ..., \mathbf{s}_i^{(m)}\}$; (3) the visible region of $\mathbf{r}_i$ is represented by a set of vectors $\{\mathbf{s}_i^{(1)} - \mathbf{r}_i, ..., \mathbf{s}_i^{(m)} - \mathbf{r}_i\}$, as called volumetric shape image (VSI) [8]. The algorithm requires an extra computation of the rough medial axis to map the boundary surface points onto the medial axis. In addition, it does not calculate the actual volume of visible region, but using a set of vectors instead. In a similar way to [8], Varanasi and Boyer [42], [43] implemented a shape segmentation algorithm. In fact, the strategies in [8], [42], [43] are related to skeleton extraction. In contrast, our method calculates the real volume of visible region without any medial axis or skeleton extraction procedure. The volume-oriented measure presented in this paper can also be considered as a complement of the existing part-aware surface measure.

2.3 Contributions

Our main contributions can be summarized as follows.

- A novel geometric measure, called the visible internal volume (VIV) function, is presented for 3D articulated shapes, which considers the volumetric context inside the shape. We first build a visibility graph between all pairs of boundary points within the shape, and then the VIV value at each boundary point is estimated by summing the volume of all visible pyramids as seen from the point.
- A new junction-aware shape descriptor for 3D articulated shapes is proposed by quantifying the VIV variation. The descriptor is computed as the Gaussian-weighted average of the VIV variation of neighborhood points of each boundary point.
- We discuss how the VIV function and the new descriptor can be incorporated into several potential applications such as initial junction extraction, shape segmentation and shape retrieval. We present a number of examples that show visually the appealing results from using our method.
- Several advantages of the presented descriptor contain that it is robust to articulation and it reflects the shape structure and deformation well without any explicit shape decomposition or prior skeleton extraction procedure. In addition, our method only depends on the geometric representation of an individual model without any reference models.

The rest of this article is organized as follows. Section 3 introduces some basic concepts and gives an overview of our algorithm. Section 4 presents the new VIV function and its computation algorithm. Section 5 proposes the new junction-aware shape descriptor. Section 6 gives the experimental results, comparison, applications and discussions. Finally, Section 7 concludes the paper.

3 PRELIMINARIES

This section introduces some basic concepts for articulated shapes and gives an overview of our algorithm.

3.1 Articulated shapes

Definition 1 (Articulated shape): *An articulated shape O consisting of K disjoint rigid parts R_1,...,R_K and L flexible junctions J_1,...,J_L, such that*

$$O = (R_1 \cup ... \cup R_K) \cup (J_1 \cup ... \cup J_L). \tag{1}$$

Intuitively, O is an articulated object if it satisfies the following conditions:

(1) *O can be decomposed into several rigid parts connected by junctions.*
(2) *The volume of each junction connected by rigid parts is relatively small when compared to its connected parts.*
(3) *Let Φ be a transformation that changes the pose of an object O. Φ is roughly considered as an articulated transformation if the transformation of any part of O is rigid (rotation and translation only) and the transformation of junctions can be non-rigid or flexible.*
(4) *The new shape O' achieved from articulation of O is again an articulated object and can articulate back to O.*

Several similar definitions also appeared in other literature [2], [5], [44]. Based on the above intuitions, some remarks should be mentioned below [3].

- Each rigid part R_i is connected and closed, $R_i \bigcap R_j = \emptyset$, $i \neq j, 1 \leq i, j \leq K$.
- Each junction J_i connects two or more rigid parts. In particular, the size of J_i can be measured by $\mathrm{vol}(J_i) \leq \varepsilon$, where $\mathrm{vol}(J_i)$ denotes the volume of J_i. $\varepsilon \geq 0$ is very small compared to the size of the rigid parts connected. A special case is $\varepsilon = 0$, which means that all junctions degenerate to single points and O is called an *ideal articulated object*.
- The articulated transformation Φ from an articulated object O to another articulated object O' is a one-to-one continuous mapping [2], [3]. This preserves the topology between the articulated parts. In particular, each deformed junction still has the volume less than or equal to ε.

The input model in this paper is considered a closed manifold triangular mesh.

Fig. 2. Illustrating the procedure of our algorithm. (a) The input mesh of a cup model. (b) Building the visibility graph (green lines) between all pairs of vertices (red points). (c) Finding the visible region (green pyramids) of each vertex based on the graph. (d) Approximating the VIV function. (e) Computing the junction-aware shape descriptor. In the color map shown in this paper, colors indicate the values of VIV or shape descriptors – from red (high) to blue (low).

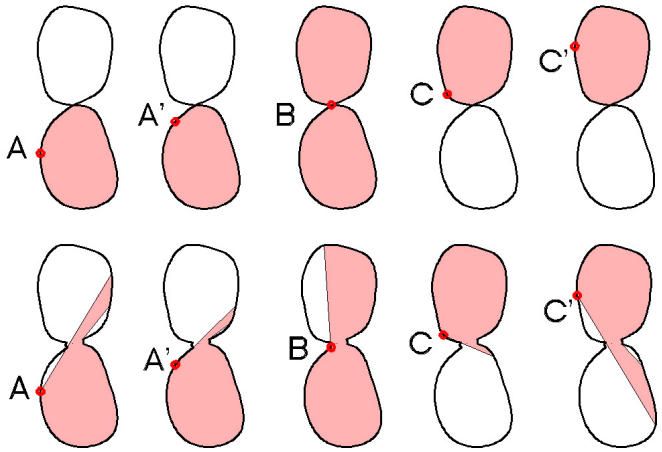


Fig. 1. Illustration of the visible region (light red) as observed from a boundary point travelling along the articulated shapes. The top row shows an ideal articulated object case, where the visible region of the junction point B undergoes a radical increase compared with the points A/A' and C/C' on the two rigid parts. The bottom row shows that, in a general articulated object, the visible region remains relatively stable or only changes gradually when moving along a rigid part. In contrast, there is always a large change in the volume (area in 2D) of visible regions as the observed point crosses junction portions.

3.2 Algorithm overview

Our method is mainly motivated from an observation that the visible regions as observed within a rigid part remain stable, but often suffer a large change in the volume when moving across the junction portion. This leads to the realization that junctions can be captured by considering occlusion or visibility inside the shape. Fig. 1 illustrates such observation by an example, where there is always a large change in the volume of visible regions within the shape as the observed point crosses junction portions.

Starting with a triangular mesh model O as an input, the main procedure of our algorithm consists of the following two steps, as illustrated in Fig. 2.

1. Calculate the visible internal volume (VIV) function for the input shape O on its mesh surface.
   1.1. First, build a visibility graph between all pairs of mesh vertices within O (see Fig. 2(b)).
   1.2. Then, the visible region as seen from each vertex is found based on the visibility graph (see Fig. 2(c)).
   1.3. Next, approximate the VIV function of O (see Fig. 2(d)), where the VIV value of each mesh vertex is estimated by summing the volumes of all the pyramids centered at this vertex and with a visible triangle of mesh as a base.
2. Compute the junction-aware shape descriptor of O by quantifying the VIV variation of neighborhood vertices of each vertex (see Fig. 2(e)).

The following sections describe each step in detail.

4 Visible Internal Volume

The first step of our algorithm is to define and calculate the values of visible regions for the input shape O. The *visible region* [42] of a point $\mathbf{p} \in O$ can be defined as a domain (connected open set), in which the inner distance [3], [4] between $\mathbf{p}$ and each point of the domain is equal to the Euclidean distance. It can be thought of as the domain spanned by shooting rays from $\mathbf{p}$ in all directions towards the surface. To quantify the visible region, we define a new geometric measure for assisting junction analysis as follows.

Definition 2 (Visible Internal Volume (VIV)): *Given a 3D shape O, the VIV at an arbitrary point $\mathbf{p} \in O$ is defined as the volume of visible region within the shape O as observed from $\mathbf{p}$.*

In the following, there are some notes for VIV that need to be mentioned first.

- The VIV is a non-negative scalar function. In this paper, we denote the VIV at $\mathbf{p} \in O$ by $f(\mathbf{p})$ and $f(O) = \{f(\mathbf{p}) : \mathbf{p} \in O\}$ for short.
- We are interested in 3D shapes defined by their boundaries, hence only the boundary points are used as landmark points for computing the VIV in our application. In the remainder of this paper, O will denote a closed triangular mesh. The VIV values are defined on all the vertices of mesh, and

linearly interpolated within the edges and triangles of mesh.
- Unlike a 'part' or 'junction', the VIV can be clearly defined as long as the visible facets of O as observed from $\mathbf{p}$ are found. Especially for the triangular mesh O, $f(\mathbf{p})$ can be computed by summing the volumes of all pyramids centered at $\mathbf{p}$ and with a full (or partial) visible triangle of O as a base.

There are several appealing features for the VIV when dealing with 3D articulated shapes. All the features are directly derived from Definition 2.

Proposition 1: *The VIV in Definition 2 is invariant to rigid transformation (translation and rotation only). When a shape undergoes scale variation, the value of its associated VIV scales accordingly.*

Proof. First we consider transformation of translation and rotation. Denote O' as the 3D shape transformed after rotating and translating a 3D shape O. Suppose for a contradiction that $f(\mathbf{p}')$ is different with $f(\mathbf{p})$, where $\mathbf{p}' \in O'$ is the transformed point corresponding to its source point $\mathbf{p} \in O$. This means that the visible region of $\mathbf{p}'$ will be inconsistent with the visible region of $\mathbf{p}$. Assume that $\mathbf{q}' \in O'$ (its source point is $\mathbf{q} \in O$) is a point on the inconsistent region, i.e. that $\mathbf{q}'$ is on the visible region of $\mathbf{p}'$ while $\mathbf{q}$ is not on the visible region of $\mathbf{p}$. That means that $\mathbf{q}'$ can be seen from $\mathbf{p}'$ within O', whereas $\mathbf{q}$ cannot be seen from $\mathbf{p}$ within O. This contradicts rigid transformation of O, since any rigid transformation is a one-to-one continuous mapping that preserves the topology between O and O'.

Then we consider scale transformation. As we mentioned before, for the triangular mesh O, $f(\mathbf{p})$ can be computed by summing the volumes of all the pyramids centered at $\mathbf{p}$ and with a full (or partial) visible triangle (or part) of the mesh O as a base. When O undergoes scale variation, each pyramid scales accordingly, resulting in that the value of the associated $f(\mathbf{p})$ scales accordingly. This completes the proof. □

To create a VIV function which is also scale independent, we can divide its value by the maximum one of all VIV values based on Proposition 1, i.e.

$$f(\mathbf{p}) = \frac{f(\mathbf{p})}{\max_{\mathbf{x}\in O}\{f(\mathbf{x})\}}. \quad (2)$$

Furthermore, being similar to the observation in [8], [42], the VIV doesn't change much within a rigid part of a shape, but changes drastically across the junction (see Fig. 1). Since the VIV computation is done over the mesh volume, it reflects the volumetric context of the shape. Being different with [8], that implicitly considers the visible volumes of interior points sampled on the medial axis, we explicitly calculate the visible volumes (i.e. Definition 2) of points on the boundary surface.

Comparison with curvature and the shape diameter

There are some inherent connections and differences between curvature, the shape diameter and the VIV.

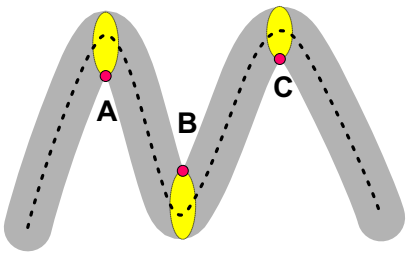


Fig. 3. Illustration for comparison with curvature and the shape diameter in a 2D case.

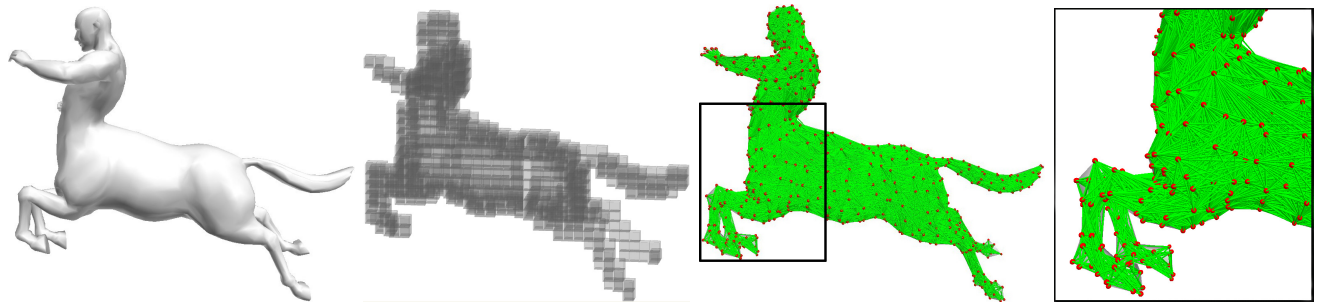
Curvature. Although curvature remains stable in each rigid part during the skeletal articulation, it cannot capture the essential difference between the points in junction regions and those in rigid parts. Curvature measures the degree of concavity and convexity, but purely from a local surface perspective. Beside, curvature may undergo an unexpected change for a point in a non-rigid junction region. The negative minimal curvature [8], [27] may help to identify some portions of the part boundaries (e.g. points **A**, **B** and **C** in Fig. 3), but it is still hard to identify the junction region (yellow region) as a whole. For the VIV definition, we can intuitively consider the curvature to be related to the '*viewing angle*' of visible region from the viewpoint.

Shape diameter. The shape diameter [17], [30], [31] essentially reflects the '*thickness*' between a boundary point to its opposite side surface. Compared with curvature, the shape diameter captures the shape volumetric information and the global structure better. For the junction detection purpose, an alternative assumption is that the junction portion should have the relatively small shape diameter. However, the small shape diameter does not necessarily lead to a junction. As exemplified by the above Fig. 3, the shape diameters along the most portion of the contour are the same, while the three points **A**, **B** and **C** reach the maximum shape diameter. For the VIV definition, we can intuitively consider the shape diameter to be related to the '*viewing depth*' of visible region from the viewpoint.

In general, compared with curvature and the shape diameter, the VIV function intuitively gives a more comprehensive measure for the volumetric context of shape. We can roughly consider the VIV as a feature combination of both the viewing angle (curvature) and viewing depth (the shape diameter) of visible region from the viewpoint located on the surface boundary. More experimental comparisons with curvature and the shape diameter will be given in Section in 6.1.

4.1 The visibility graph

We utilize the visibility graph for estimating the VIV function, where the visibility graph is constructed by the links connecting the visible vertices of O. For our purpose, we define the visibility graph between all pairs of vertices as follows.



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

Fig. 4. Building the visibility graph between all pairs of vertices on the mesh. (a) The original model. (b) An octree of depth 5. (c) The visibility graph (green lines) between all pairs of vertices (red points). (d) The magnified view.

Definition 3 (Visibility graph): *Given a closed manifold triangular mesh O, its vertex visibility graph G is defined as follows. The nodes of G correspond to the vertices of the mesh, and two vertices are connected by an arc of G if and only if they are mutually visible within O, i.e. if the line segment connecting the two vertices is in the interior or along the boundary of O.*

The problem of computing the visibility graph of 2D polygons has been widely studied in computational geometry [45], which can be computed in $O(n^2)$ time [46]. But, unfortunately, the 3D visibility graph for polyhedra is much harder. The visibility graph is also associated with the problem of finding the shortest visible path between two points within a closed space, which is NP-hard in 3D [45]. Several recent studies [3], [4] have also explored the visibility graph-based algorithms for computing inner distances of articulated shapes. Instead of computing the *exact* visibility graph in 3D space, that is a complex and expensive task, we present an approximation algorithm for computing roughly the visibility graph of triangular meshes based on octree acceleration.

4.1.1 Octree construction

A crucial part for defining the visibility graph is to check the visibility between all pairs of vertices within O. This operation can be accelerated using a spatial search structure, such as octree and kd-tree. In our preprocessing step, an axis-aligned octree is typically built around the boundary mesh for accelerating the visibility graph computation.

Starting with the bounding box of O, we subdivide it into 8 equally sized cells. Each cell is recursively split into 8 children cells as long as it contains the triangles of the mesh. The iteration is terminated until a user-chosen depth d (typically 4 or 5) is reached. This process is similar to octree construction in [47]. After constructing the octree, we classify its cells into three types: *boundary cells* intersecting with the boundary mesh of O, *interior cells* inside O and *exterior cells* outside O. The algorithm of cell classification is referred to Ref. [47], [48]. Fig. 4(b) shows an octree on the centaur model.

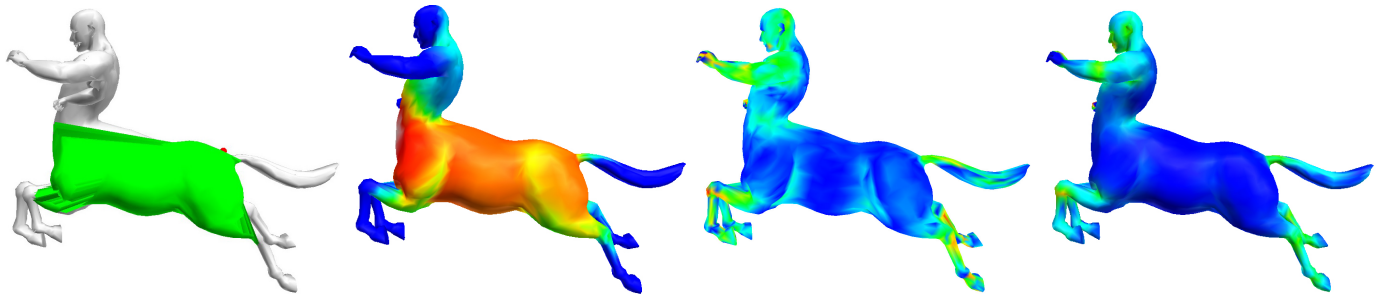
4.1.2 Visibility checking

Following Definition 3, we define the visibility graph G over all vertices of O by connecting each pair of vertices $\mathbf{p}_i$ and $\mathbf{p}_j$. If the line segment (denoted by $\boldsymbol{\ell}$) connecting $\mathbf{p}_i$ and $\mathbf{p}_j$ falls entirely within O, $\mathbf{p}_i$ and $\mathbf{p}_j$ are visible and $\boldsymbol{\ell}$ is added to the graph G. Note that if $\mathbf{p}_i$ and $\mathbf{p}_j$ are on the same triangle of the mesh, they are marked as visible and linked with an arc in the visibility graph; otherwise, we collect the intersection points between $\boldsymbol{\ell}$ and the boundary mesh of O, with acceleration of the octree's boundary cells. The intersection algorithm is based on Ref. [46]. In particular, if $\boldsymbol{\ell}$ and the mesh only intersect at the two endpoints (i.e. $\mathbf{p}_i$ and $\mathbf{p}_j$) of $\boldsymbol{\ell}$, $\mathbf{p}_i$ and $\mathbf{p}_j$ are either visible ($\boldsymbol{\ell}$ lies inside O) or invisible ($\boldsymbol{\ell}$ lies outside O). To distinguish inside or outside, one may choose the midpoint of $\boldsymbol{\ell}$ and decide if the midpoint is inside the given O using the classical point-in-polyhedron algorithm [46], e.g. the well-known ray-casting algorithm. To avoid extra intersection computation between the mesh and the ray through the midpoint, we use the classified cell types of octree to speed up. This is done by simply finding the cell containing the midpoint, and then checking its cell type as follows:

(1) If the cell type is interior, $\mathbf{p}_i$ and $\mathbf{p}_j$ are visible.
(2) If the cell type is exterior, $\mathbf{p}_i$ and $\mathbf{p}_j$ are invisible.
(3) If the cell type is boundary, the point-in-polyhedron algorithm is further used for checking.

Note that if $\boldsymbol{\ell}$ intersects the mesh at multiple points except $\mathbf{p}_i$ and $\mathbf{p}_j$, we simply mark it as invisible. Although such a simplified processing may not be accurate in some cases, such as that $\mathbf{p}_i$ and $\mathbf{p}_j$ both are on the same one side of a 3D cube model, it is relatively uncommon for 3D articulated models. A more accurate and robust classification of intersection points can also be applied to our work such as considering some degeneracies of $\boldsymbol{\ell}$ tangent to the mesh surface, but this leads to an increase in computation time and complexity. Fig. 4(c) and Fig. 4(d) show the approximation visibility graph between all pairs of vertices using our algorithm.

In general, octree construction and cell classification do not take more than ten seconds even for large meshes, and consequently, building the visibility graph even on



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

Fig. 5. Illustration of the visible region, VIV and junction-aware shape descriptor. (a) shows the visible region (green pyramids) at a vertex on the centaur's body. (b) visualizes the VIV function, where there is a great change on junctions. (c) shows the VIV variation. (d) visualizes the junction-aware shape descriptor, where junctions are highlighted by the descriptor. Warmer colors (red and yellow) show high values and cooler colors (green and blue) show low values.

large meshes takes only a few minutes (see Table 1 for some examples). In contrast, if there is no acceleration of spatial octree, it often takes tens of minutes or even one hour for building the visibility graph of large meshes.

4.2 The VIV computation

After building the visibility graph G, the next work is to find the visible region of each vertex based on G, and then compute its VIV. Given a specific vertex $\mathbf{p} \in O$, we first search all triangles of the mesh O and determine whether each triangle $T_i \in O$ is visible from the viewpoint $\mathbf{p}$. Benefiting from the visibility graph G computed before, the visibility test for triangle T_i can be performed fast, by evaluating the visibility between $\mathbf{p}$ and the three vertices of T_i. Base on G, we roughly define the visibility of T_i from $\mathbf{p}$ as the following three cases:

(1) If the three vertices of T_i are all visible from $\mathbf{p}$, T_i is marked as the *full visible triangle*.
(2) If one or two vertices of T_i are visible from $\mathbf{p}$, T_i is marked as the *partial visible triangle*.
(3) Otherwise, T_i is marked as the *invisible triangle*.

Fig. 5(a) shows the visible region at a selected vertex, by connecting the selected vertex and its full/partial visible triangles into the pyramids. Note that we only use the visibility graph G for approximately deciding the visibility of triangles. Computing the exact visible region for complex polyhedral shapes is quite involved, but it is not necessary for our purpose. Consequently, we define the VIV function of each vertex $\mathbf{p}$ by

$$f(\mathbf{p}) = \sum_{T_j \in O} \alpha(\mathbf{p}, T_j) \mathrm{vol}(\mathbf{p}, T_j), \tag{3}$$

where $\mathrm{vol}(\mathbf{p}, T_j)$ is the volume of the jth-pyramid centered at $\mathbf{p}$ and with the jth-triangle T_j as a base. For the reader's convenience, we give the expression of the volume of a pyramid in the following equation [49], where $\mathbf{q}_j^1$, $\mathbf{q}_j^2$ and $\mathbf{q}_j^3$ are the three vertices of T_j:

$$\mathrm{vol}(\mathbf{p}, T_j) = \frac{1}{6}(\mathbf{p} - \mathbf{g}_j) \cdot \mathbf{N}_j,$$

where $\mathbf{g}_j = (\mathbf{q}_j^1 + \mathbf{q}_j^2 + \mathbf{q}_j^3)/3$ and $\mathbf{N}_j = (\mathbf{q}_j^2 - \mathbf{q}_j^1) \wedge (\mathbf{q}_j^3 - \mathbf{q}_j^1)$.

The weight α in Eq. (3) is defined by

$$\alpha(\mathbf{p}, T_j) = \frac{\mathrm{vis}(T_j)}{3},$$

where $\mathrm{vis}(T_j)$ is the number of visible vertices of T_j from $\mathbf{p}$ (it is directly obtained from G), i.e. α = 1, 2/3, 1/3 or 0. Alternatively, the scale independent VIV function can be computed using Eq. (2).

Fig. 5(b) visualizes the VIV function of the centaur model, where the VIV value of each triangle of O is linearly interpolated by its three vertices. Fig. 6 displays the VIV values of series of 3D articulated shapes. Observe that there is a great change on the junction portions.

Computational complexity

Let n be the number of vertices on the input triangular mesh O and m be the number of triangles on O ($m \approx 2n$ for triangular meshes). First, it takes time $O(m)$ to check whether the line segment between one pair of vertices is inside O, where checking intersections between the line segment and all the triangles is $O(m)$ and the inside-outside testing using the octree's cell types is $O(1)$. As a result, the complexity of constructing the visibility graph G between all pairs of vertices is $O(n^2 m)$ (or $O(n^3)$). After G is ready, computing the VIV value of each vertex in Eq. (3) is linear. Together with $O(n^3)$ construction of the visibility graph, an upper bound of running time for the VIV computation takes $O(n^3)$.

5 Junction-aware shape descriptor

The second step of our algorithm aims to develop a junction-aware shape descriptor for a 3D articulated model based on its VIV function. For each vertex $\mathbf{p} \in O$, we first find its k-nearest neighborhood using the ANN library that can be found at: http://www.cs.umd.edu/~mount/ANN/. One can consider several distances to define the neighborhood, such as the Euclidean distance or geodesic distance. We have tried both and found that the Euclidean distance gives the similar results to

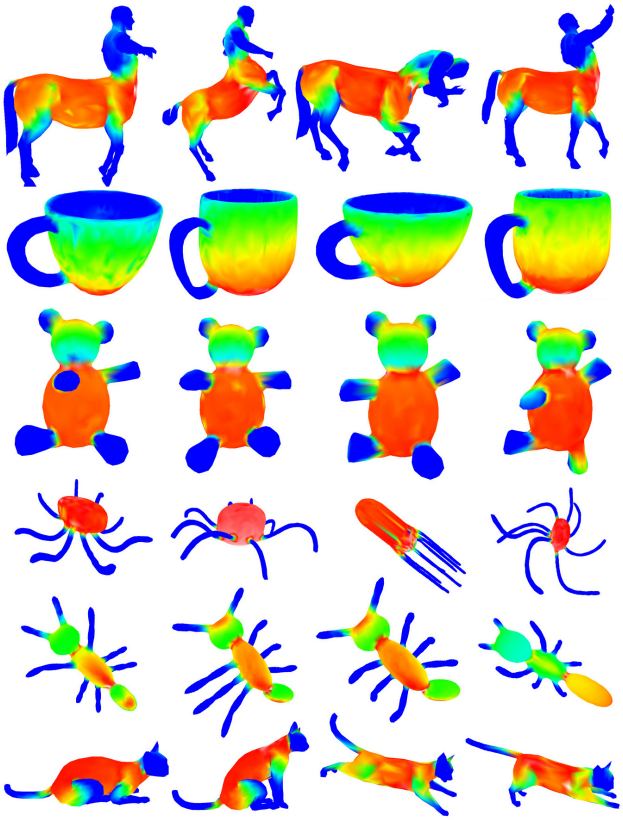


Fig. 6. Visualizing the VIV function of series of 3D articulated shapes. Rows are centaur, cup, teddy, octopus, ant and cat. Columns show the various poses.

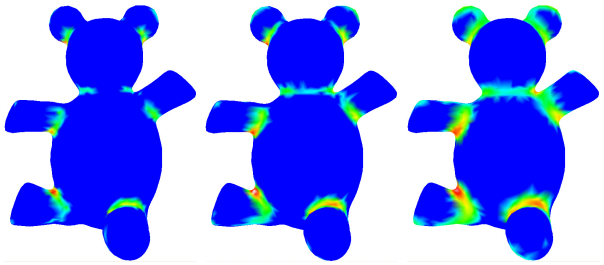
geodesic distance, but can be computed faster. Let $N_k(\mathbf{p})$ denotes the set of k-nearest neighborhood vertices of $\mathbf{p}$.

Then we define the junction-aware shape descriptor at each vertex $\mathbf{p}$ as the Gaussian-weighted average of the VIV variation between $\mathbf{p}$ and its neighboring vertices:

$$\mathcal{F}(\mathbf{p}) = \frac{\sum\limits_{\mathbf{x}\in N_k(\mathbf{p})} |f(\mathbf{x}) - f(\mathbf{p})| \exp[-\|\mathbf{x}-\mathbf{p}\|^2/(2\sigma^2)]}{\sum\limits_{\mathbf{x}\in N_k(\mathbf{p})} \exp[-\|\mathbf{x}-\mathbf{p}\|^2/(2\sigma^2)]}, \quad (4)$$

where $\mathcal{F}(\mathbf{p})$ denotes the shape descriptor of $\mathbf{p}$, $f(\mathbf{p})$ is the VIV function value of $\mathbf{p}$ in Eq. (3), $|f(\mathbf{x})-f(\mathbf{p})|$ is the VIV variation between $\mathbf{p}$ and its neighborhood vertex $\mathbf{x}$, and σ is the standard deviation of the Gaussian filter. In our implementation, σ for each vertex $\mathbf{p}$ is adaptively computed as the average of distances between $\mathbf{p}$ and its neighboring vertices. $\mathcal{F}(\mathbf{p})$ is insensitive to shape deformation that does not alter the volumetric shape locally, which includes articulated deformation or skeleton based movements or piecewise rigid transformation.

The following Algorithm 1, named **VertexDescriptor**, gives the pseudo-code for applying the above Eq. (4) to a single vertex $\mathbf{p}$, where the algorithm returns the junction-aware descriptor value. Fig. 5(c) shows the VIV variation on the mesh by computing the average of difference of the VIV between each vertex and its neighborhood vertices. Fig. 5(d) visualizes the values of junction-aware



(a) (b) (c)

Fig. 7. The junction-aware shape descriptor is relative to the number k of neighborhood vertices. (a) $k = 8$. (c) $k = 15$. (d) $k = 30$. The small k sharpens the junction features, while the large k over-smooths the junction features.

shape descriptor by applying the Gaussian filter to the VIV variation. Observe that the junction portions are highlighted by the descriptor.

Algorithm 1 VertexDescriptor(Vertex $\mathbf{p}$)

```
Find k-nearest neighborhood {q_i} of p;
k = |{q_i}|;
Compute σ as the average of Euclidean distances between {q_i} and p;
normalizer = 0;
sum = 0;
for i := 1 to k do
    d = ||p − q_i||;
    W = exp(−d²/(2σ²));
    variation = |f(p) − f(q_i)|;
    sum += W·variation;
    normalizer += W;
end for
return descriptor = sum/normalizer;
```

Parameters. Algorithm 1 includes two parameters: k and σ. The descriptor with the small neighborhood number k highlights the *thin* junction features, while the descriptor with the large k identifies the *thick* junction features. In Fig. 7(a), the small k sharpens the junction features, but this may result a discontinuous feature (see Teddy's neck). In contrast, in Fig. 7(c), the large k over-smooths the junction features, but this may introduce the redundant features (see Teddy's neck too). In our implementation we typically set $k = 15$ for large meshes. Note that, in Eq. (4), we are assuming a cut-off for the Gaussian filter at a distance σ that penalizes the neighborhood vertices far from $\mathbf{p}$. In practice, one may choose a large σ, such as the maximal distances between $\mathbf{p}$ and its neighboring vertices, but this often over-smooths the junction features.

Noise and outlier. Noisy data of the mesh surface may lead to unstable VIV computation, which will produce an undesirable shape descriptor. To overcome this, we pre-smooth the VIV function on the noisy mesh by averaging the VIV value of each vertex with its 1-ring

neighborhood. After smoothing the VIV function, Algorithm 1 is recalled for computing the descriptor using the smooth VIV function. In addition, to avoid the local perturbation and obtain a more distinguishable junction-aware descriptor, we also eliminate the influence of some small or outlier's descriptor values through setting the smallest 10% values of descriptor as zero, where the percent of outliers can be chosen by the user.

6 Experimental Results

We have implemented our algorithm in C++ and experimented with a large number of articulated models. The tested models are selected from several well-known 3D articulated shape databases, such as Princeton Segmentation Benchmark (PSB) [16] and ISDB [17]. The experimental results and some potential applications are presented here for demonstrating the effectiveness of our method.

All the experiments were run on a PC with a 2.60 GHz processor and 8 GB memory, excluding the time of loading models. An axis-aligned octree at depth 5 is constructed in the preprocessing step and this procedure usually takes less than a few seconds. Table 1 summarizes the time in seconds for some articulated models referred to in this paper, where '#Verts' is the number of mesh's vertices, '#Faces' is the number of mesh's triangles, 'T1' is the time for octree construction and cell classification at depth 5, and 'T2' is the time for building the visibility graph between all pairs of vertices and computing the shape descriptor. A great deal of the running time is spent in building the visibility graph, while the stage of computing the VIV function and shape descriptor is almost real time. Fig. 8 shows the computation time for several models with respect to various resolutions, where we typically simplify four meshes (i.e. teddy, centaur, cup and octopus) into various resolutions from 200 to 2000 vertices. The results show that the computation time increases accordingly as the number of the mesh's vertices increases. Since our method only depends on the geometric representation of the individual shape itself, it can be computed off-line and this property is of great importance to many further applications that require fast and effective processing, such as feature recognition, shape retrieval and motion tracking.

To verify the capability of our method, several articulated shapes are tested and demonstrated in Fig. 9. The second column on the left shows the computed visibility graph that captures the shape structures well. The third column on the left displays the VIV function, where there is a great change on junctions. The rightmost column shows our descriptor that highlights the junction features in the warm colors. For all the examples in Fig. 9, the neighborhood number is set to $k = 15$, while the smallest 10% values of descriptor are regarded as outliers and discarded.

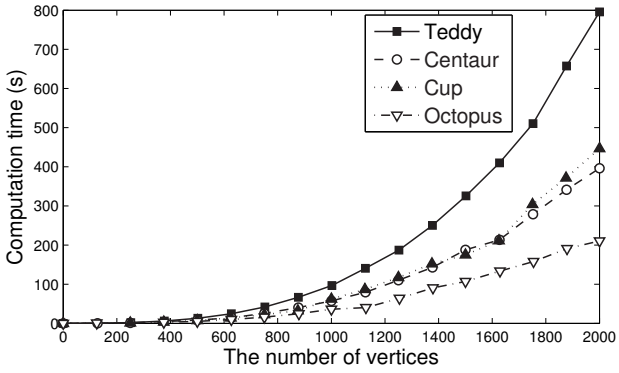


Fig. 8. Computation time for several models with respect to their various resolutions. As the number of mesh's vertices increases, computation time increases accordingly.

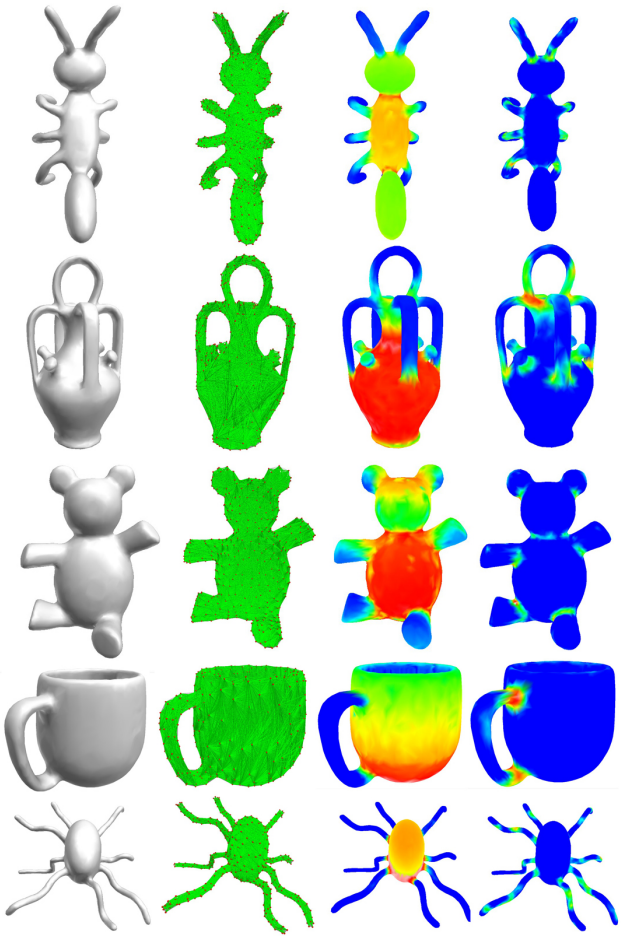


Fig. 9. Visualizing the visibility graph, VIV and junction-aware shape descriptor. Rows are ant, vase, teddy, cup and octopus. The leftmost column shows the original models, the second one shows the visibility graph, the third one visualizes the VIV function, and the rightmost one displays the junction-aware shape descriptor.

TABLE 1
Computation time.

Model	Fig.	#Verts	#Faces	T1[a](s)	T2[b](s)
Centaur	4	352	700	0.43	2.82
Ant	9	402	800	0.81	3.40
Teddy	9	752	1500	1.98	41.76
Octopus	9	1002	2000	1.15	50.84
Cup	9	1500	3000	6.17	174.89

a. T1 is the time for octree construction and cell classification.
b. T2 is the time for building the visibility graph between all pairs of vertices and computing the shape descriptor.

6.1 Comparison with several relevant methods

We also compare our work with several relevant methods including curvature, mesh saliency, and the shape diameter. Fig. 10 shows various examples of results using curvature and mesh saliency and our descriptor. The results suggest that curvature only reflects the convexities or concavities of local surface, and it is very sensitive to local perturbation and noise.

Mesh saliency [32], [33] is a measure of regional importance for 3D meshes using a center-surround operator on Gaussian-weighted mean curvatures. It is able to identify the regions that are different from their surrounding context. Although the saliency measure is superior to mesh curvature, it does not capture the volumetric context inside the shape. In addition, mesh saliency has the same drawback with curvature during identifying the junction features, i.e. that the convex and concave features on local surfaces do not completely reflect the characteristics of junctions. For example, the ears of teddy and the tips of octopus's eight arms are convex portions (see Fig. 10), but they are not junctions. In contrast, our junction-aware descriptor can distinguish the junction features well.

As we mentioned before, the SDF [17], [30], [31] approximates the double distance from each mesh vertex to the corresponding medial axis, i.e. the shape diameter. Although the SDF takes into account the interior of the shape, it does not capture the general volumetric context, as shown in Fig. 3 in a 2D case. Fig. 11 shows another 3D example. Here, Fig. 11(a) displays the SDF of an octopus model [30] and Fig. 11(b) shows a SDF-based descriptor by applying our Algorithm 1 in Section 5 to the SDF instead of our original VIV function. Fig. 11(c) and Fig. 11(d) show our VIV function and junction-aware descriptor. Note that the shape diameters of points on each arm of octopus are almost same, resulting in that the SDF-based shape descriptor cannot capture the bending changes of octopus's arms. In contrast, our descriptor captures such bending deformations well.

6.2 Applications

A number of tasks in computer vision and computer graphics can benefit from the VIV function and junction-aware shape descriptor. In this paper, we explore several potential applications of our method in initial junction extraction, shape segmentation and shape retrieval.

A direct application of our shape descriptor is for initial junction extraction. This can be simply done by using a threshold to divide the descriptor values into two regions, i.e. junction portions and rigid parts. Here we mark these vertices with their descriptor values larger than the threshold as the junction portions, and the remaining vertices are marked as the rigid parts. Fig. 12

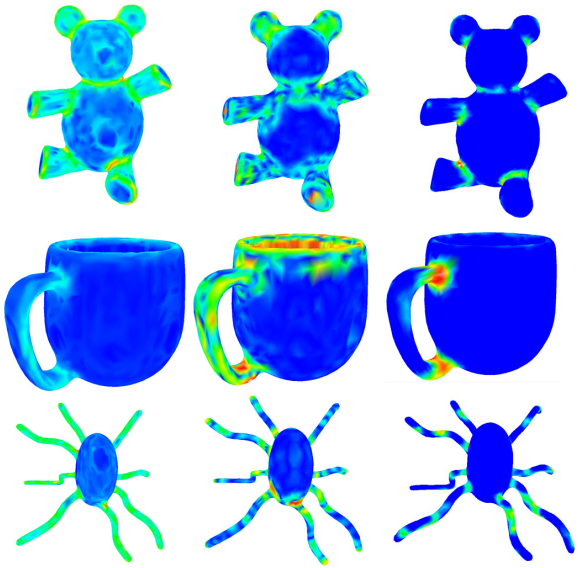
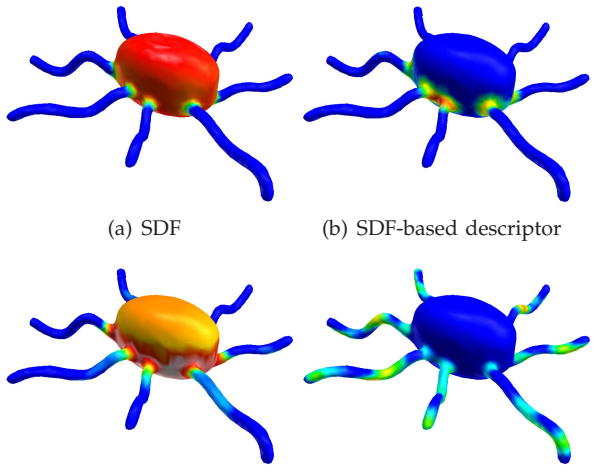


Fig. 10. Comparison of our descriptor (right column) and curvature (left column) and mesh saliency (middle column).



(a) SDF (b) SDF-based descriptor

(c) VIV (d) Junction-aware descriptor

Fig. 11. Comparison of our method and the SDF for an octopus model. Note that each arm of octopus is a bending cylinder-like shape, so the shape diameters of points on each arm of octopus are almost same. This results in that the SDF cannot capture the bending changes of octopus's arms. In contrast, our method captures such bending changes well.

Fig. 12. Application to initial junction extraction by using a threshold to divide the descriptor values into two regions, where the colored regions are identified as the initial junction portions and the remaining ones are rigid parts.

shows an example of this application for three models, where the threshold is typically selected as the median one of descriptor values of all vertices. Note that a clear-cut junction might need further post processing which leads to a future work.

Another possible application is to improve some part-aware segmentation algorithms for 3D articulated shapes [8], [30] by considering the VIV function instead of some existing measures, such as the SDF [30] or VSI [8]. For instance, the SDF-based mesh segmentation algorithm [30] is composed of two steps: (1) use soft-clustering of the mesh triangles to k clusters based on their SDF values, and (2) find the actual segmentation using k-way graph-cut to include local mesh geometric properties. Our implementation follows the above two steps, but the SDF of each triangle in the first step is replaced by our VIV function, where the VIV value of each triangle is linearly interpolated by its three vertices. Fig. 13 shows the segmentation results on the mesh surfaces of several articulated models with $k = 3$ clusters, which reveal the similar parts in all of them.

Recently, Ling and Jacobs [3] proposed an algorithm for computing and applying the inner distances for building new 2D shape descriptors. One of its central work is to first construct a visibility graph connecting all visible boundary points, and then find the shortest path in the graph as the inner distance. Following the similar way, we presented a visibility graph based algorithm for computing the inner distances of 3D articulated volumetric models [4], which extends [3] from 2D to 3D. However, the current implementation of 3D inner distance computation [4] still limits to 3D volumetric models, which has not been applied to polygonal meshes yet. A by-product of our VIV computation is 3D visibility graph between all pairs of vertices, which can be directly used for 3D inner distances (ID) computation [4] of triangular meshes. Fig. 14 shows the application to articulated shape retrieval, where the average precision-recall curves are tested on the ISDB database [17] with several known descriptors: Euclidean distance (ED), geodesic distance (GD), shape distribution (SD), spherical harmonic descriptor (SHD), and solid angle histogram (SAH) in terms of the performance in retrieving similar shapes. The retrieval results show that the ID descriptor generated by our visibility graph performs better than other descriptors for articulated models.

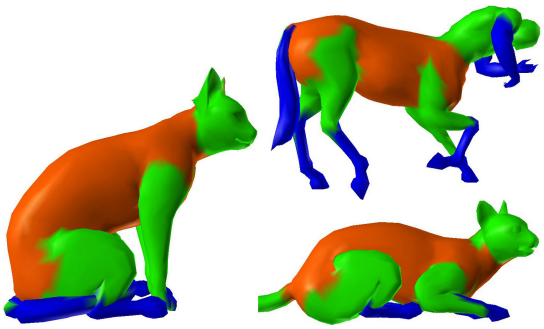


Fig. 13. Application to articulated shape segmentation, where the consistent segmentation results on mesh surfaces of several models reveal the similar parts.

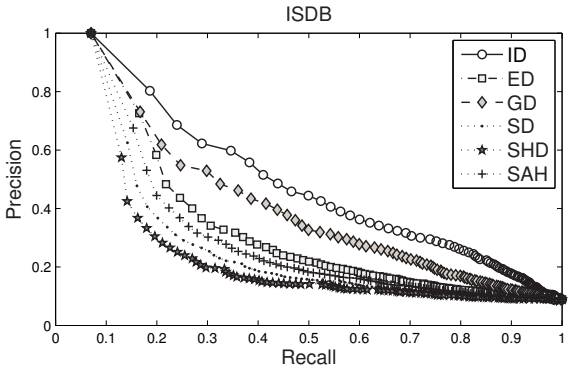


Fig. 14. Application to articulated shape retrieval. The precision-recall curves compare the retrieval results achieved with the ID descriptor versus other methods on ISDB, where the by-product visibility graph of VIV is used for computing the ID descriptor of triangular meshes.

6.3 Discussions

One issue of our method is its robustness to the object's representation. The VIV function is only meaningful on an object which defines a closed volume. This means that the object containing holes and gaps may need some pre-processing, such as mesh repairing [47] to produce a closed manifold surface. However, in the octree construction process, the vertex normal on a mesh can be used for improving cell classification [47], [48] even for some non-closed meshes, and consequently, we can produce an approximating visibility graph between all pairs of

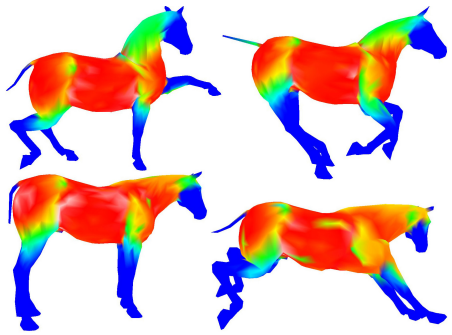


Fig. 15. Illustrating the VIV function for different poses of the same horse. Note that there is a relatively great change of VIV values on the necks of the top two horses, but there is no apparent change on the necks of the bottom two horses.

vertices with the help of the octree. Once the meaningful visibility graph is built, our VIV computation can be run without further considering the non-closed cases. This creates robustness to small cracks when handling holes in the boundary.

In this paper we deal with a single object in isolation. Consequently, the VIV function of a single shape may lack sufficient cues to identify its all junctions (see e.g. Fig. 15). It is quite difficult to resolve this issue if just using an isolate model. However, it is of interest to improving our method by referring to multiple poses of the same object, where different poses may complement each other for identifying the missing junction portions. To achieve this, we plan to borrow some recent ideas from joint-segmentation/co-segmentation of multiple poses [29]. It is, however, beyond the scope of this paper, and we will leave this study to the future work.

7 Conclusion

In this paper we consider the problem of junction-aware shape descriptor for a single 3D mesh model. The challenge is how to encode junction information on the boundary surface of a shape. By developing a novel visible internal volume (VIV) function and quantifying its variation in the neighborhood of each point on the surface, we manage to achieve such a junction-aware descriptor. The effectiveness of our method is testified by a number of examples on several well-known 3D articulated shape databases. Furthermore, we explore some potential applications including initial junction detection, shape segmentation and shape retrieval, which could benefit from our method.

The VIV function is a new powerful geometric measure associating volumetric information to the mesh surface. It offers a new perspective for understanding 3D articulated shapes. We believe that the VIV definition can further assist many other geometry processing applications such as feature recognition, segmentation, matching, motion tracking and functional prediction. In addition, our current implementation of building the visibility graph for VIV computation takes a long time for large meshes. It is possible to speed it up significantly by parallelizing on multi-core platforms, because visibility checking between each pair of vertices is completely independent from all other vertices. The parallel implementation of our algorithm is an independent future work.

Acknowledgments

The research is supported by the National Science Foundation of China (61272229, 61003095). The first author is also supported by the National Technological Support Program for the 12th-Five-Year Plan of China (2012BAJ03B07), the Chinese 863 Program (2012AA040902), and the Chinese 973 Program (2010CB328001). The third author is supported by NSFC (61203317) and the Chinese 863 Program (2012AA09A409). Prof. Paul is supported by the ANR-NSFC (60911130368).

References

[1] A. M. Bronstein, M. M. Bronstein, and R. Kimmel, *Numerical geometry of non-rigid shapes*. Springer, 2007.

[2] A. M. Bronstein, M. M. Bronstein, A. M. Bruckstein, and R. Kimmel, "Analysis of two-dimensional non-rigid shapes," *International Journal of Computer Vision*, vol. 78, no. 1, pp. 67–88, 2008.

[3] H. Ling and D. Jacobs, "Shape classification using the inner-distance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 2, pp. 286–299, 2007.

[4] Y.-S. Liu, K. Ramani, and M. Liu, "Computing the inner distances of volumetric models for articulated shape description with a visibility graph," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 12, pp. 2538–2544, 2011.

[5] C. Wang, Y.-S. Liu, M. Liu, J.-H. Yong, and J.-C. Paul., "Robust shape normalization of 3D articulated volumetric models," *Computer-Aided Design*, vol. 44, no. 12, pp. 1253–1268, 2012.

[6] A. Ion, N. M. Artner, G. Peyré, W. G. Kropatsch, and L. D. Cohen, "Matching 2D and 3D articulated shapes using the eccentricity transform," *Computer Vision and Image Understanding*, vol. 115, no. 6, pp. 817–834, 2011.

[7] G. Verbitsky, R. Nussinov, and H. Wolfson, "Flexible structural comparison allowing hinge-bending, swiveling motions," *Proteins*, vol. 34, no. 2, pp. 232–254, 1999.

[8] R. Liu, H. Zhang, A. Shamir, and D. Cohen-Or, "A part-aware surface metric for shape analysis," *Computer Graphics Forum*, vol. 28, no. 2, pp. 397–406, 2009.

[9] D. Reniers and A. Telea, "Part-type segmentation of articulated voxel-shapes using the junction rule," *Computer Graphics Forum*, vol. 27, no. 7, pp. 1845–1852, 2008.

[10] R. Su, C. Sun, and T. D. Pham, "Junction detection for linear structures based on Hessian, correlation and shape information," *Pattern Recognition*, vol. 45, no. 10, pp. 3695–3706, 2012.

[11] F. Zhao, P. Mendonca, R. Bhotika, and J. Miller, "Model-based junction detection algorithm with applications to lung nodule detection," in *ISBI'07*, 2007, pp. 504–507.

[12] M. Shatsky, R. Nussinov, and H. J. Wolfson, "Flexible protein alignment and hinge detection," *Proteins*, vol. 48, no. 2, pp. 242–256, 2002.

[13] T. Shibuya, "Fast hinge detection algorithms for flexible protein structures," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 7, no. 2, pp. 333–341, 2010.

[14] U. Emekil, D. Schneidman-Duhovny, H. Wolfson, R. Nussinov, and T. Haliloglu, "Hingeprot: automated prediction of hinges in protein structures," *Proteins*, vol. 70, no. 4, pp. 1219–1227, 2008.

[15] S. C. Flores and M. B. Gerstein, "Flexoracle: predicting flexible hinges by identification of stable domains," *BMC Bioinformatics*, vol. 8, no. 215, 2007.

[16] X. Chen, A. Golovinskiy, and T. Funkhouser, "A benchmark for 3D mesh segmentation," *ACM Transactions on Graphics*, vol. 28, no. 3, p. Article 73, 2009.

[17] R. Gal, A. Shamir, and D. Cohen-Or, "Pose-oblivious shape signature," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 2, pp. 261–271, 2007.

[18] L. Parida, D. Geiger, and R. Hummel, "Junctions: Detection, classification, and reconstruction," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 7, pp. 687–698, 1998.

[19] R. Elias and R. Laganière, "JUDOCA: JUnction Detection Operator based on Circumferential Anchors," *IEEE Transactions on Image Processing*, vol. 21, no. 4, pp. 2109–2118, 2012.

[20] W. Yu, K. Daniilidis, and G. Sommer, "Rotated wedge averaging method for junction characterization," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'98)*, 1998, pp. 390–395.

[21] M. A. Ruzon and C. Tomasi, "Edge, junction, and corner detection using color distributions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 11, pp. 1281–1295, 2001.

[22] R. Laganière and R. Elias, "The detection of junction features in images," in *International Conference on Acoustic, Speech and Signal processing*, 2004, pp. 573–576.

[23] F. Deschenes and D. Ziou, "Detection of line junctions in gray-level images," in *International Conference on Pattern Recognition (ICPR'00)*, vol. 3, 2000, pp. 754–757.

[24] W. Barrett and K. Petersen, "Houghing the Hough: peak collection for detection of corners, junctions and line intersections," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'01)*, vol. II, 2001, pp. II–302–II–309.

[25] M. A. Cazorla and F. Escolano, "Two bayesian methods for junction classification," *IEEE Transactions on Image Processing*, vol. 12, no. 3, pp. 317–327, 2003.

[26] M. Hilaga, Y. Shinagawa, T. Kohmura, and T. Kunii, "Topology matching for fully automatic similarity estimation of 3D shapes," in *Proceedings of ACM SIGGRAPH*, 2001, pp. 203–212.

[27] O. K.-C. Au, C.-L. Tai, H.-K. Chu, D. Cohen-Or, and T.-Y. Lee, "Skeleton extraction by mesh contraction," *ACM Transactions on Graphics*, vol. 27, no. 3, p. Article 44, 2008.

[28] T. Ju, M. Baker, and W. Chiu, "Computing a family of skeletons of volumetric models for shape description," *Computer-Aided Design*, vol. 39, no. 5, pp. 352–360, 2007.

[29] Q. Huang, V. Koltun, and L. Guibas, "Joint shape segmentation with linear programming," *ACM Transactions on Graphics*, vol. 30, no. 6, p. Article 125, 2011.

[30] L. Shapira, A. Shamir, and D. Cohen-Or, "Consistent mesh partitioning and skeletonization using the shape diameter function," *The Visual Computer*, vol. 24, no. 4, pp. 249–259, 2008.

[31] L. Shapira, S. Shalom, A. Shamir, R. H. Zhang, and D. Cohen-Or, "Contextual part analogies in 3D objects," *International Journal of Computer Vision*, vol. 89, no. 2–3, pp. 309–326, 2010.

[32] C.-H. Lee, A. Varshney, and D. Jacobs, "Mesh saliency," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 659–666, 2005.

[33] Y.-S. Liu, M. Liu, D. Kihara, and K. Ramani, "Salient critical points for meshes," in *ACM Symposium on Solid and Physical Modeling (SPM'07)*, 2007, pp. 277–282.

[34] P. Shilane and T. Funkhouser, "Distinctive regions of 3D surfaces," *ACM Transactions on Graphics*, vol. 26, no. 2, p. Article 7, 2007.

[35] T. Funkhouser and P. Shilane, "Partial matching of 3D shapes with priority-driven search," in *Symposium on Geometry Processing (SGP'06)*, 2006, pp. 131–142.

[36] U. Castellani, M. Cristani, and V. Murino, "Statistical 3D shape analysis by local generative descriptors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 12, pp. 2555–2560, 2011.

[37] H. Tabia, "A new 3D-matching method of nonrigid and partially similar models using curve analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 4, pp. 852–858, 2011.

[38] Y. Fang, Y.-S. Liu, and K. Ramani, "Three dimensional shape comparison of flexible protein using the local-diameter descriptor," *BMC Structural Biology*, vol. 9, no. 29, pp. 1–15, 2009.

[39] Y.-S. Liu, Y. Fang, and K. Ramani, "IDSS: deformation invariant signatures for molecular shape comparison," *BMC Bioinformatics*, vol. 10, no. 157, pp. 1–14, 2009.

[40] Y.-S. Liu, M. Wang, J.-C. Paul, and K. Ramani, "3DMolNavi: A web-based retrieval and navigation tool for flexible molecular shape comparison," *BMC Bioinformatics*, vol. 13, no. 95, pp. 1–7, 2012.

[41] Y.-S. Liu, Q. Li, G.-Q. Zheng, K. Ramani, and W. Benjamin, "Using diffusion distances for flexible molecular shape comparison," *BMC Bioinformatics*, vol. 11, no. 480, pp. 1–15, 2010.

[42] K. Varanasi, "Spatio-temporal modeling of dynamic 3D scenes from visual data," Ph.D. dissertation, Computer Science and Applied Mathematics at the Universite de Grenoble, France, 2010.

[43] K. Varanasi and E. Boyer, "Temporally coherent segmentation of 3D reconstructions," in *International Conference on 3D Data Processing, Visualization and Transmission (3DPVT)*, 2010.

[44] R. Gopalan, P. Turaga, and R. Chellappa, "Articulation-invariant representation of non-planar shapes," in *ECCV'10*, 2010, pp. 286–299.

[45] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *"Chapter 15: Visibility Graphs", Computational Geometry: Algorithms and Applications*, 3rd ed. Springer, 2008.

[46] J. O'Rourke, *Computational Geometry in C*, 2nd ed. Cambridge University Press, 1998.

[47] T. Ju, "Robust repair of polygonal models," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 888–895, 2004.

[48] B. Adams and P. Dutré, "Interactive boolean operations on surfel-bounded solids," in *Proceedings of SIGGRAPH'03*, 2003, pp. 651–656.

[49] M. Desbrun, M. Meyer, P. Schröder, , and A. Barr, "Implicit fairing of irregular meshes using diffusion and curvature flow," in *Proceedings of ACM SIGGRAPH*, 1999, pp. 317–324.