Neurocomputing 215 (2016) 32–47

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

VIV: Using visible internal volume to compute junction-aware shape descriptor of 3D articulated models

Yu-Shen Liu [a,b,c,*], Hongchen Deng [a], Min Liu [d], Lianjie Gong [a]

[a] School of Software, Tsinghua University, Beijing, China
[b] Key Laboratory for Information System Security, Ministry of Education of China, China
[c] Tsinghua National Laboratory for Information Science and Technology, China
[d] Department of Mechanical Engineering, Tsinghua University, China

ARTICLE INFO

Article history:
Received 5 January 2015
Received in revised form
20 June 2015
Accepted 27 June 2015
Available online 10 June 2016

Keywords:
3D articulated shapes
Junction detection
Shape descriptor
Visible internal volume (VIV)
Visibility graph

ABSTRACT

An articulated shape is composed of a set of rigid parts connected by some flexible junctions. Junctions have been demonstrated to be critical local features 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 the 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.

1. Introduction

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* [1,3–6], which assumes that the non-rigid shape consists a set of rigid parts connected by flexible junctions (or named *joints*/*hinges* [1,7]). Each of rigid parts has a certain degree of freedom to move, and junctions are relatively small compared to the parts they connect.

Junctions, as critical local features, provide valuable local information for analyzing 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, detecting junctions and parts can help to improve the performance of articulated shape segmentation [8,9]. Many medical visual tasks such as vessel tracking may rely on junction extraction, which can facilitate diagnosis and understanding of pulmonary vascular diseases [10]. In molecular structure analysis, many flexible molecules can be viewed as 3D articulated objects with special hinge/junction sites [7,11], where identifying such feature sites is a fundamental problem for protein functional prediction and comparison. Although the abilities to infer junction information of 3D articulated shapes have proven to be useful in many applications, efficient description and detection of junctions still remain a research challenge due to the high complexity of 3D articulated deformation.

To address this issue, this paper presents a 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 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 VIV for shape description is that it is robust to articulation and can reflect shape structure and deformation well without any

* Corresponding author at: School of Software, Tsinghua University, Beijing 100084, China. URL: http://cgcad.thss.tsinghua.edu.cn/liuyushen/
E-mail addresses: liuyushen@tsinghua.edu.cn (Y.-S. Liu), denghc09@gmail.com (H. Deng), minliu@tsinghua.edu.cn (M. Liu), lianjiegong@gmail.com (L. Gong).

http://dx.doi.org/10.1016/j.neucom.2015.06.115
0925-2312/

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) [12] and ISDB [13]. Some experimental results and applications are presented for demonstrating the effectiveness of our method.

Note that a class of junction detection methods (e.g. [7,11,14]) 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

During the past decades, many junction detection methods in 2D images were proposed in the literature. Junctions in 2D can be classified by the number of wedges [15], including two-junctions (e.g. corners), three-junctions (e.g. *T* or *Y*-junctions), four-junctions (e.g. *X*-junctions), and so on. Such junction features are very important for image analysis and object recognition [15,16]. *Junction detectors/descriptors* do the job of describing and identifying both locations and information of junctions [16]. There are two basic approaches for detecting junction features in 2D: *region-based* approaches and *edge-based* ones. The former usually models junctions as image regions where some wedge-shaped regions meet [10,15], and the later often describes junctions as image points where two or more ridges meet [16,17]. Generally speaking, both region-based and edge-based approaches utilize some characteristics which are usually available only in 2D images for detecting junctions. Such 2D characteristics are generally difficult to be obtained by only analyzing 3D geometric shapes. A survey of junction detection in 2D images is beyond the scope of this paper.

Although junction detectors/descriptors have been broadly studied in 2D images, very few papers explicitly discussed this issue for 3D shapes.

2.1. Junction detection for 3D shapes

In general, the existing approaches for junction detection are strongly dependent on 3D shape representation in specific applications. There are several possible ways to solve this issue. The first category of approaches is fitting-based methods. For example, Zhao et al. [18] introduced a fitting-based technique for detecting vessel junctions of 3D CT data, where the parametric junction model is fitted using three tori patches and an ellipsoid surface. However, it is not an easy task to fit the junction region only using simple quadratic surfaces for complex 3D shapes.

The second category of approaches is based on structure analysis methods, which are generally used in protein structure analysis. Specially, if given a pair of protein structures [7,11,14], the junctions are often found through structure alignment between the two proteins. If only an individual protein structure is given without referring to any other proteins [19,20], junction prediction usually takes advantage of some additional chemical information of the protein itself, such as amino acid sequences. In contrast, our method is designed to treat the geometric representation of an individual 3D shape, while there is no prior structure information or non-geometric property required for the shape.

The third category of approaches is skeleton-based methods. Many techniques in computer graphics focus on topology or graph extraction of 3D shapes with surface representations (e.g. polygonal meshes [21–23]) or volumetric representations [23]. The topology extraction process produces a thin skeleton (or medial axis), possibly with additional junctions and branches corresponding to the logical components of a given shape [21,22]. Nevertheless, the extracted skeleton is often very sensitive to perturbation and noise on the shape's boundary, where robust skeleton computation is not an easy task. In addition, the computed skeleton only induces a rough junction position on the curve-skeleton itself, while the final junction region corresponding to the boundary surface still needs to be computed and refined. 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 prior skeleton extraction procedure.

The fourth category of approaches is segmentation-based methods. 3D shape segmentation/partition methods have gained much attention in recent years, but most of methods have to face the same question of how to define parts or part boundaries of 3D objects [8,24–26]. The classical part-type segmentation techniques aim to analyze the geometric structure of an individual shape in order to detect its parts or part boundaries (e.g. [8,9,25]). Recently, another trend of methods is to segment the shapes jointly, which utilizes the features from multiple similar shapes to improve segmentation of each shape [24,26]. In essence, there is also a strong connection between part-partitioning and skeletonizing [9,26]. For example, 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. In contrast, 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.

The fifth category of approaches is based on shape descriptor methods. Our method also falls into this category. A shape descriptor is a concise representation of the shape that expresses some of its specific properties [4,13]. A simple realization of junction detection can be based on measuring the local surface properties of the shape boundary, such as local surface curvature, point signatures on surface [27], dihedral angles and gradient [28]. However, such measures are limited by their local surface nature. Recently, there are some efforts to develop new shape descriptors for detecting the interesting and important surface regions or features, such as salient regions [29,30] and distinctive regions of surfaces [31,32]. However, they are still purely surface-based and do not take account of volumetric information within 3D shapes.

It is worthwhile to mention the *shape-diameter* function (SDF) (or named *local-diameter* descriptor) presented by Shamir et al. [13,25,26]. The SDF is a scalar function defined on the boundary surface of a 3D shape, which provides an implicit link between the local volume of the shape and its boundary surface. Intuitively, the local shape diameter is an approximated sphere diameter measured from the medial axis to a point on the boundary surface, which is computed as follows. For each point $\mathbf{p}$ on the boundary surface, the algorithm first 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 weighted average length of all the intersection rays is calculated as the shape diameter of $\mathbf{p}$. Our previous work has applied the SDF to flexible protein shape comparison [33]. 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 the junction features of 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 the characteristics of junctions. A reasonable junction-aware descriptor should also capture the volumetric context of 3D shapes. Recently, as a volume-oriented geometric metric, the inner distance [3] was developed, 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 described a visibility graph-based algorithm for computing inner distances of 3D volumetric shapes [4], which also derived some applications [34–36]. Although the inner distance is articulation-invariant, it needs to compute a pair-wise distance and is not junction-aware. In contrast, this paper develops a new volume-oriented measure (i.e. VIV) and quantifies its variation as a junction-aware shape descriptor, which is not a pair-wise distance measure compared with inner distance.

Another recent work related to ours is Ref. [8], in which a part-aware surface measure was developed by considering the volumetric context inside a 3D shape. It is based on an observation that the visible region as observed inside a shape's volume provides a strong hint for part transition. The visible region generally remains stable within a part, while often 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 $\{\mathbf{s}_i^{(1)}, \ldots, \mathbf{s}_i^{(m)}\}$; (3) the visible region of $\mathbf{r}_i$ is implicitly represented by a set of vectors $\{\mathbf{s}_i^{(1)} - \mathbf{r}_i, \ldots, \mathbf{s}_i^{(m)} - \mathbf{r}_i\}$, 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 explicitly provide the volume of visible region, but only using a set of vectors instead. In essence, [8] is a pair-wise surface metric similar to inner distance, which measures the difference between the VSI vectors of two points on the surface as the shape descriptor. In a similar way to [8], Varanasi and Boyer [37,38] implemented a shape segmentation algorithm. In fact, the strategies in [8,37,38] are related to skeleton extraction. In contrast, our method explicitly calculates the actual volume of visible region without any medial axis or skeleton extraction procedure. Furthermore, being different with [8], our junction-aware descriptor is not a pair-wise distance measure. The volume-oriented VIV measure presented in this paper can be also considered as a complement of existing part-aware surface measures, which will be demonstrated in Section 6.2.

2.2. Contributions

Our main contributions can be summarized below.

- A novel geometric measure, called 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 junction-aware shape descriptor for 3D articulated models is proposed by quantifying the VIV variation. The descriptor is computed as the Gaussian-weighted average of the VIV variation of neighborhood points for each boundary point.
- We discuss how the VIV function and the new descriptor can be incorporated into several potential applications such as junction extraction, shape segmentation and shape retrieval.

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. In addition, our method only depends on the geometric representation of an individual model without any reference models.

The remainder 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 consists of K disjoint rigid parts $R_1,\ldots,R_K$ and L flexible junctions $J_1,\ldots J_L$, such that

$$O = (R_1 \cup \cdots \cup R_K) \cup (J_1 \cup \cdots \cup J_L). \quad (1)$$

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 is relatively small compared to its connected rigid 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 [1,5,39]. Based on the above intuitions, some remarks should be mentioned below [3].

- Each rigid part R_i is connected and closed, $R_i \bigcap R_j = \varnothing$, $i \neq j$, $1 \leq i, j \leq K$.
- 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 connected rigid parts. 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 [3,1]. This preserves the topology between the articulated parts. In particular, each deformed junction still has the volume less than or equal to ε.

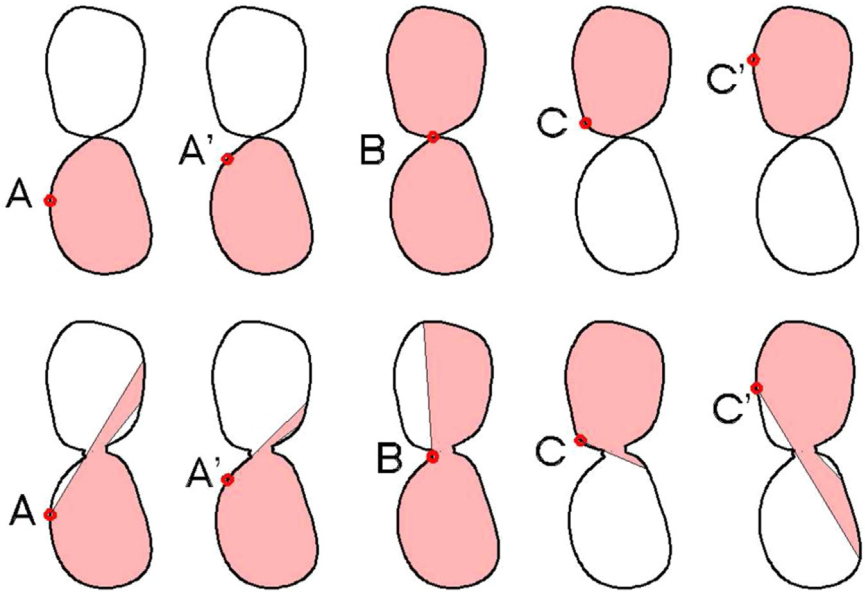


Fig. 1. Illustration of the visible region (light red) as observed from a boundary point traveling along the articulated shapes. The top row shows an ideal articulated object case, where the visible region of the junction point B undergoes a radical increase compared with the points A/A' and C/C' on 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. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

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

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 from 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

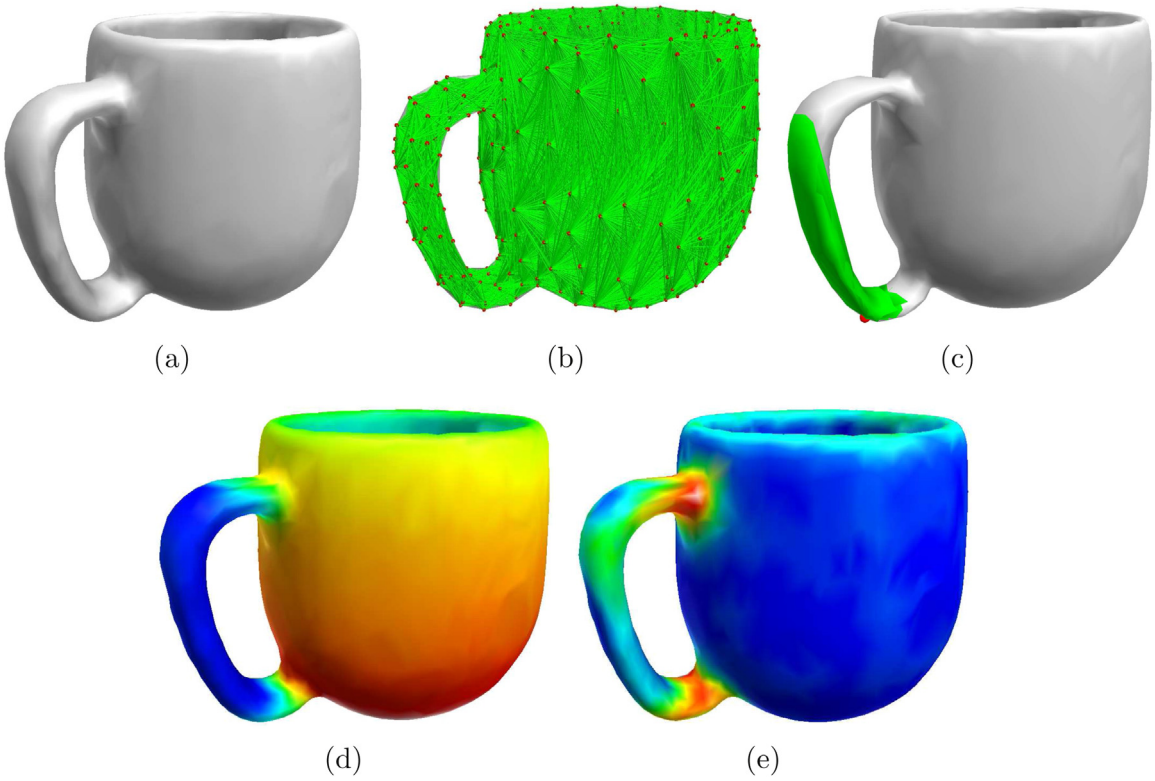


Fig. 2. Illustrating the procedure of our algorithm. (a) The input mesh of a cup model. (b) Building the visibility graph (green lines) between all pairs of vertices (red points). (c) Finding the visible region (green pyramids) of each vertex based on the graph. (d) Approximating the VIV function. (e) 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). (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

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. 2b).
   1.2. Then, the visible region as seen from each vertex is found based on the visibility graph (see Fig. 2c).
   1.3. Next, approximate the VIV function of O (see Fig. 2d), 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. 2e).

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 the visible regions for an input shape O. The *visible region* [37] 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 in 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.

1. 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.
2. 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 a mesh, and linearly interpolated within the edges and triangles of the mesh.
3. 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 translational and rotational transformation. Denote O' as the 3D shape transformed after rotating or 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.

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

Furthermore, being similar to the observation in [8,37], the VIV does not 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.

4.1. Comparison with curvature and shape diameter

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

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 [22,8] 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 at a point $\mathbf{p}$ to be related to the '*viewing angle*' of visible region from the viewpoint $\mathbf{p}$.

Shape diameter: The shape diameter [13,25,26] 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 in 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 $\mathbf{p}$.

In general, compared with curvature and shape diameter, the VIV function gives a more comprehensive measure for the volumetric context of a shape. We can roughly consider the VIV as a feature combination of both the viewing angle (curvature) and

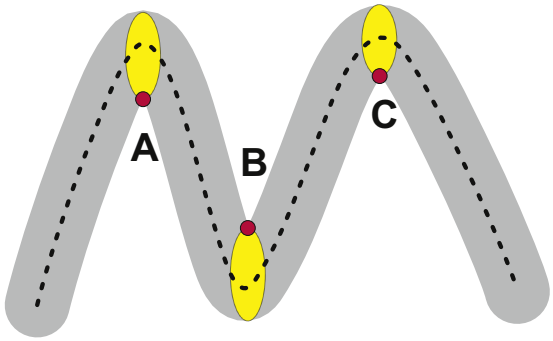


Fig. 3. Illustration for comparison with curvature and the shape diameter in a 2D case. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

viewing depth (the shape diameter) of the visible region from a viewpoint located on the surface boundary. More experimental comparisons with curvature and the shape diameter will be given in Section in 6.1.

4.2. 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 pair of vertices of O. For our purpose, we define the visibility graph between all pairs of vertices as follows.

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 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 [40], which can be computed in $O(n^2)$ time [41]. 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 [40]. 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.

4.2.1. Octree construction

A crucial part for defining the visibility graph is to check the visibility between all pairs of vertices within O. This calculation 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 [42]. 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 Refs. [42,43]. Fig. 4b shows an octree of a centaur model.

4.2.2. 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 $\mathcal{L}$) connecting $\mathbf{p}_i$ and $\mathbf{p}_j$ falls entirely within O, $\mathbf{p}_i$ and $\mathbf{p}_j$ are visible and $\mathcal{L}$ 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 $\mathcal{L}$ and the boundary mesh of O, with acceleration of the octree's boundary cells. The intersection algorithm is based on Ref. [41]. In particular, if $\mathcal{L}$ and the mesh only intersect at the two endpoints (i.e. $\mathbf{p}_i$ and $\mathbf{p}_j$) of $\mathcal{L}$, $\mathbf{p}_i$ and $\mathbf{p}_j$ are either visible ($\mathcal{L}$ lies inside O) or invisible ($\mathcal{L}$ lies outside O). To distinguish inside or outside, one may choose the midpoint of $\mathcal{L}$ and decide if the midpoint is inside the given O using the classical point-in-polyhedron algorithm [41], 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 $\mathcal{L}$ 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 $\mathbf{p}_j$ are both on the same one side of a 3D cube model, which is relatively uncommon for 3D articulated models. A more accurate and robust classification of intersection points can also be applied to our work, for instance, by considering some degeneracies of $\mathcal{L}$ tangent to the mesh surface. But this accurate classification leads to an increase in computation time and complexity. Fig. 4c and Fig. 4d show the approximated visibility graph between all pairs of vertices using our approximation 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 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 for a large mesh.

4.3. 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. Based 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. 5a shows the visible region at a selected vertex, by connecting the selected vertex and its full/partial visible triangles into

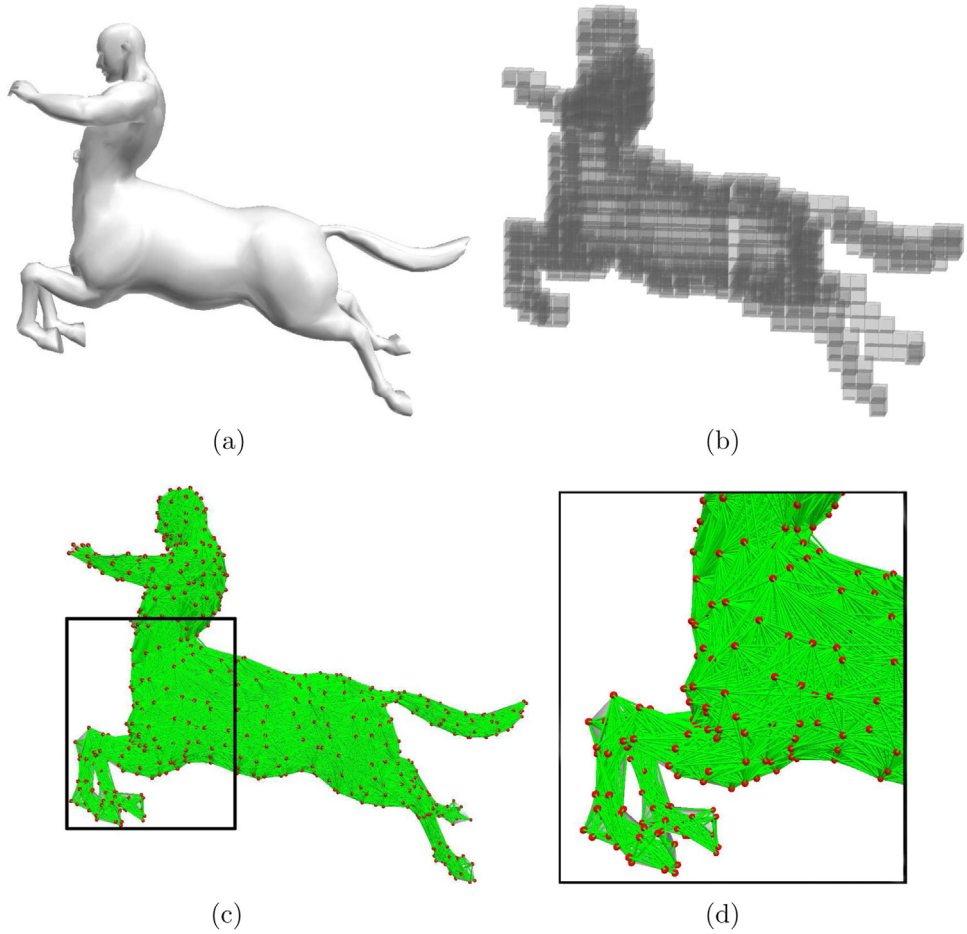


Fig. 4. Building the visibility graph between all pairs of vertices on the mesh. (a) The original model. (b) An octree of depth 5. (c) The visibility graph (green lines) between all pairs of vertices (red points). (d) The magnified view. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

Table 1
Computation time.

Model	Figure	#Vertices	#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.

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 the base. For the reader's convenience, we give the expression of the volume of a pyramid in the following equation [44], 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. $\alpha = 1, 2/3, 1/3$ 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 in the junction portions.

4.4. 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.

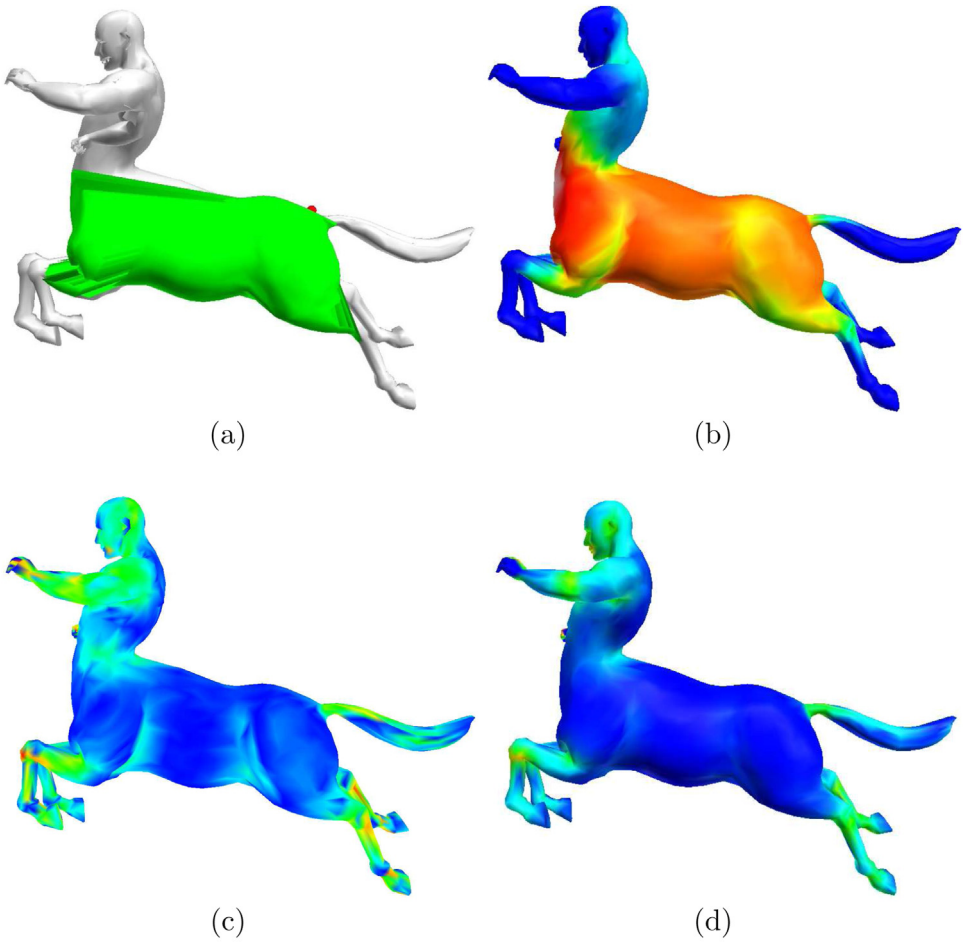


Fig. 5. Illustration of the visible region, VIV and junction-aware shape descriptor. (a) shows the visible region (green pyramids) at a vertex on the centaur's body. (b) visualizes the VIV function, where there is a great change on junctions. (c) shows the VIV variation. (d) visualizes the junction-aware 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. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

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

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

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 neighboring 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. 5c shows the VIV variation on the mesh by computing the average of difference of the VIV between each vertex and its neighborhood vertices. Fig. 5d visualizes the values of junction-aware 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 $\mathbf{p}$).

1: Find k-nearest neighbors $\{\mathbf{q}_i\}$ of $\mathbf{p}$;
2: $k = |\{\mathbf{q}_i\}|$;
3: Compute σ as the average of Euclidean distances between $\{\mathbf{q}_i\}$ and $\mathbf{p}$;
4: normalizer=0;
5: sum=0;
6: **for** i:=1 to k **do**
7: $d = \|\mathbf{p} - \mathbf{q}_i\|$;
8: $W = \exp(-d^2/(2\sigma^2))$;
9: variation=$|f(\mathbf{p}) - f(\mathbf{q}_i)|$;

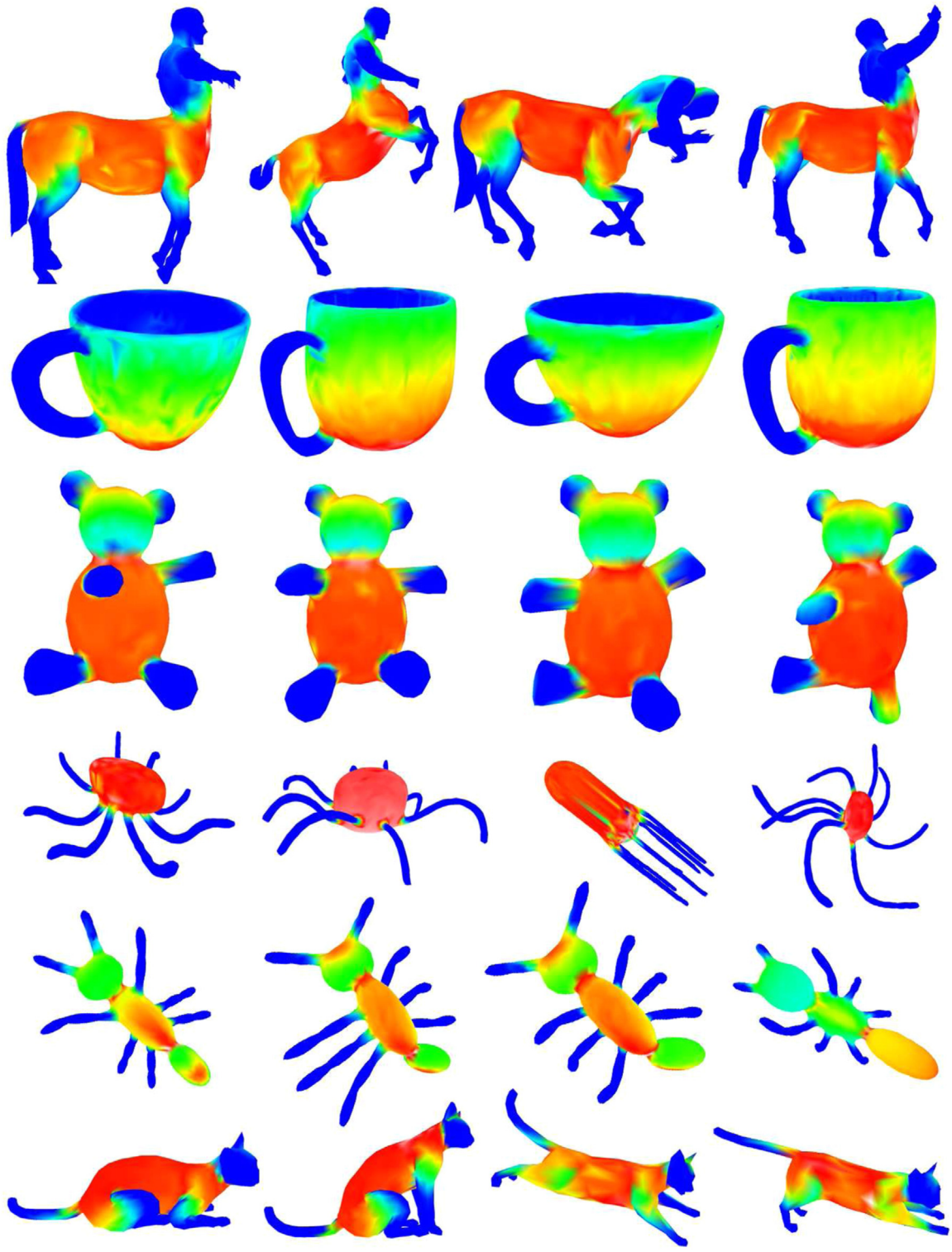


Fig. 6. Visualizing the VIV function of series of 3D articulated shapes. Rows are centaur, cup, teddy, octopus, ant and cat. Columns show the various poses.

10: sum $+=W\cdot$variation;
11: normalizer $+=W$;
12: **end for**
13: **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. 7a, the smaller k sharpens the junction features, but this may result a discontinuous feature (see Teddy's neck). In contrast, in Fig. 7c, the larger 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 **p**. In practice, one may

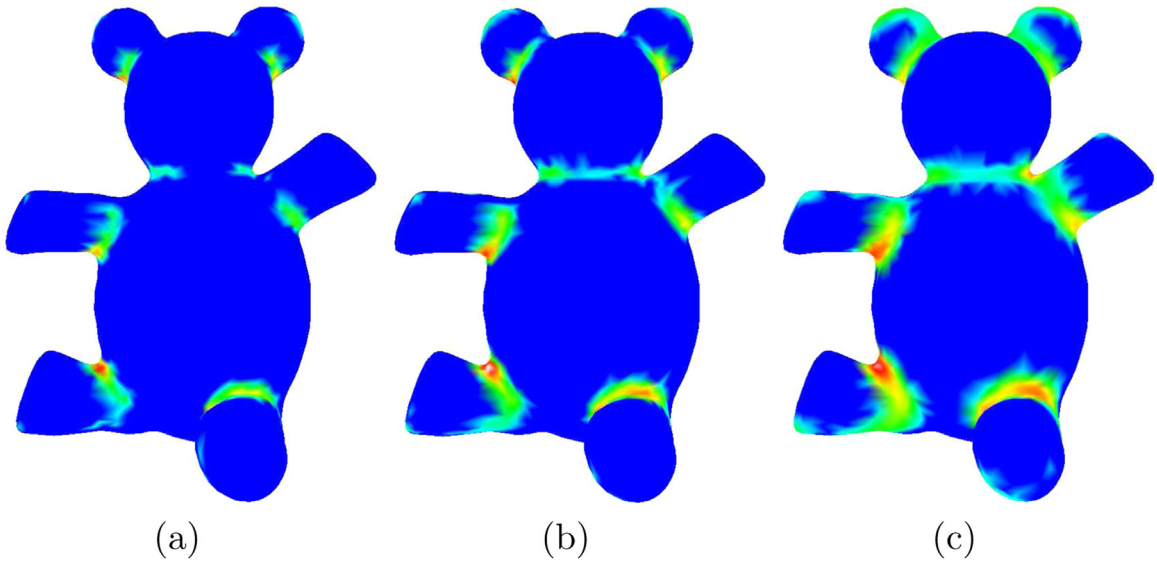


Fig. 7. The junction-aware shape descriptor is relative to the number k of neighboring vertices. (a) $k=8$. (c) $k=15$. (d) $k=30$. The smaller k sharpens the junction features, while the larger k over-smooths the junction features.

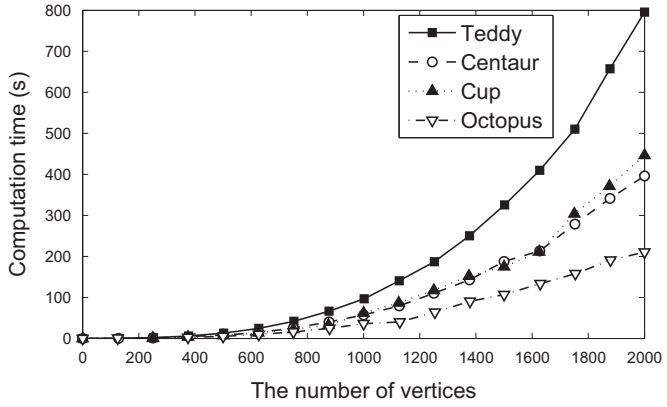


Fig. 8. Computation time for several models with respect to their various resolutions. As the number of mesh's vertices increases, computation time increases accordingly.

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 conduct a pre-smoothing on the VIV function for a noisy mesh by averaging the VIV value of each vertex within 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 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) [12] and ISDB [13]. The experimental results and some potential applications are presented here for demonstrating the effectiveness of our proposed 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 '#Vertices' 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 drastic 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 are discarded.

6.1. Comparison with several relevant methods

We also compared our work with several relevant methods including curvature, mesh saliency, and shape diameter. Fig. 10 shows various examples of results using curvature and mesh saliency and our descriptor. The results suggest that curvature only

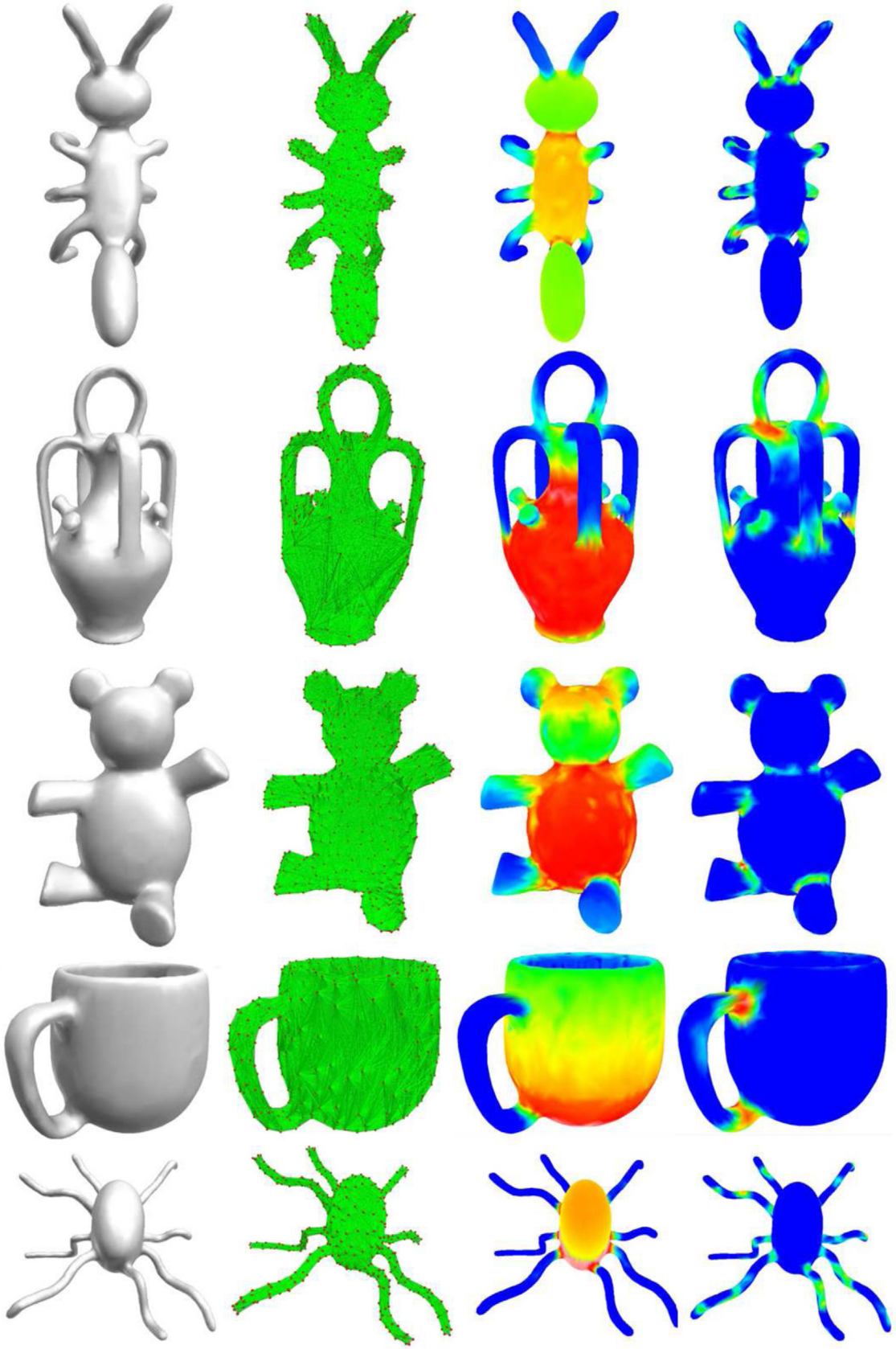


Fig. 9. Visualizing the visibility graph, VIV and junction-aware shape descriptor. Rows are ant, vase, teddy, cup and octopus. The leftmost column shows the original models, the second one shows the visibility graph, the third one visualizes the VIV function, and the rightmost one displays the junction-aware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

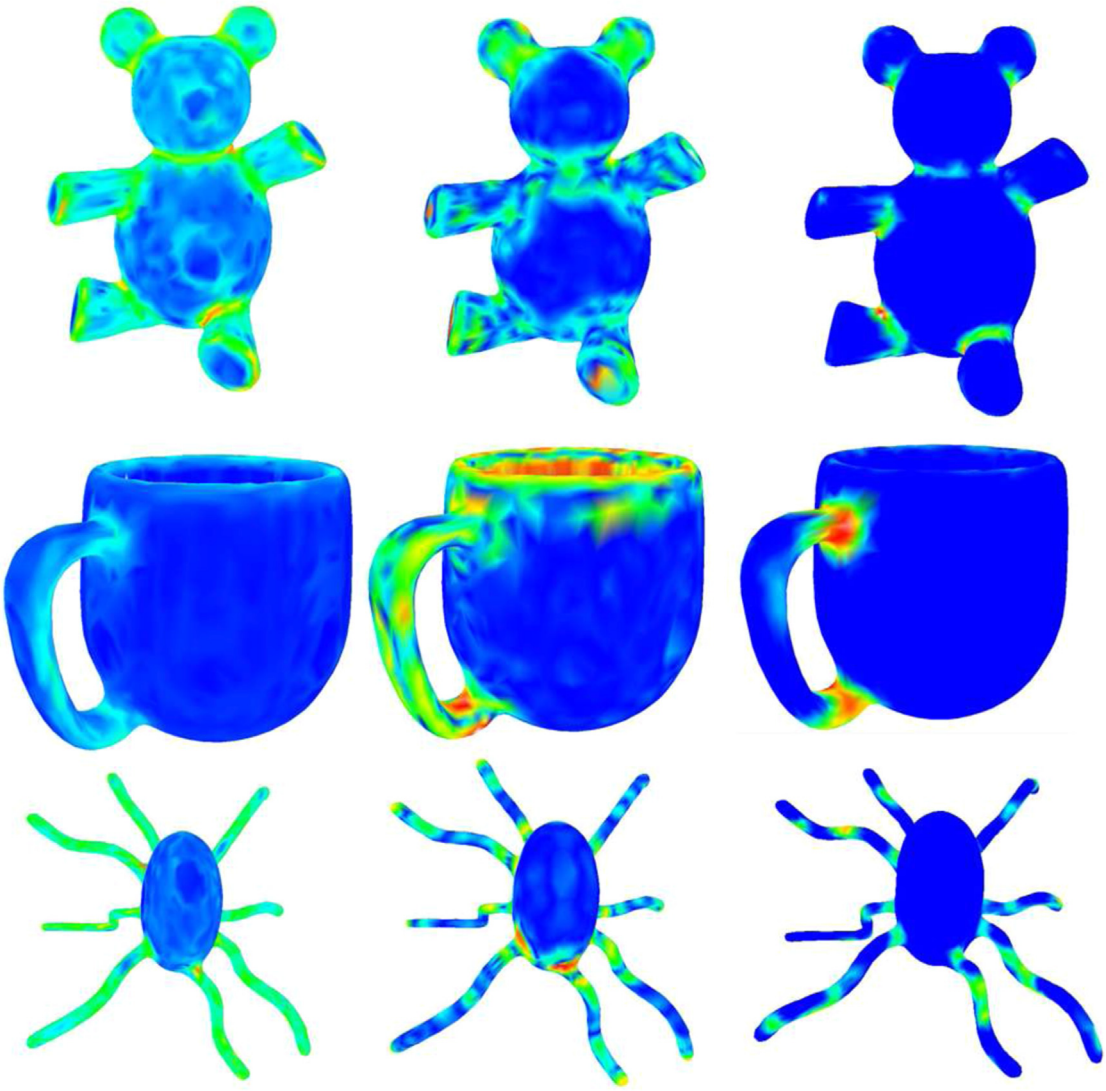


Fig. 10. Comparison of our descriptor (right column) with curvature (left column) and mesh saliency (middle column).

reflects the convexities or concavities of local surface, and it is very sensitive to local perturbations and noises.

Mesh saliency [29,30] 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 [13,25,26] 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. 11a displays the SDF [25] of a model of table lamp with an adjustable folding arm, and Fig. 11b shows a SDF-based descriptor by applying our Algorithm 1 in Section 5 to SDF instead of our original VIV function [45]. Fig. 11c and Fig. 11d show our VIV function and junction-aware descriptor. Note that the shape diameters of points on the lamp's arm are almost same, resulting in that the SDF-based descriptor cannot capture the joint features of lamp's arm. In contrast, our descriptor captures such joint features well.

6.2. Applications

Some potential applications in computer vision and computer graphics may benefit from the VIV function and the junction-aware shape descriptor. 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 of vertices into two types: junction regions and rigid parts. Here we mark these vertices with their descriptor values larger than the threshold as the candidate junction portions, and the remaining vertices are marked as the rigid parts. Fig. 12 illustrates this application for several models, where the threshold is typically selected as the median one of descriptor values of all vertices and the junction regions of each model are highlighted by orange color.

Based on the initial junction regions extracted, we can further compute the *handle loop* [46,47] which is a critical geometric

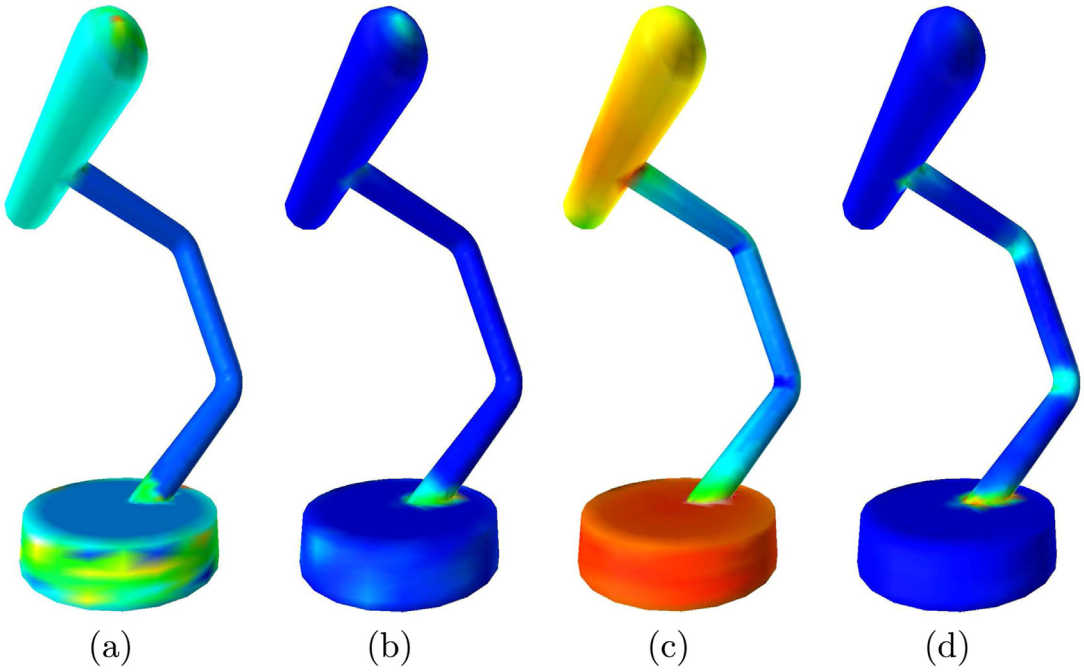


Fig. 11. Comparison of our method and the SDF, where a model of table lamp with an adjustable folding arm is selected for test. (a) SDF. (b) SDF-based descriptor [45]. (c) VIV. (d) Junction-aware descriptor. Note that the lamp arm is a bending cylinder-like shape with two joints, so the shape diameters of points on the lamp's arm are almost same. This results in that the SDF cannot capture the joint features of lamp's arm (see (b)). In contrast, our descriptor captures such joint features well (see (d)).

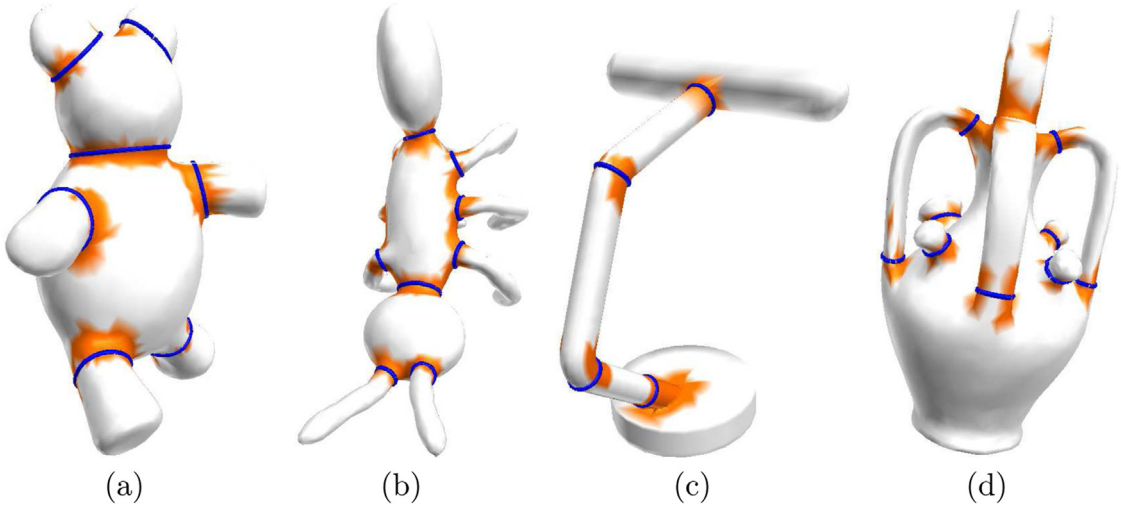


Fig. 12. Application to junction extraction (colored regions) and handle loop approximation (blue curves). (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

feature. The algorithm of handle loop approximation has been described in our previous work [45]. Being different from [45], this paper uses VIV-based shape descriptor for approximating the handle loops instead of SDF-based shape descriptor in [45]. However, since the algorithm of handle loop approximation is not a contribution to this article, we just summarize the main process of the algorithm. It consists of three steps as follows.

(1) Firstly, all the vertices on the initial junction regions are grouped into different clusters, one of which denotes a junction region to be further refined.
(2) Then, the vertices on each clustering junction are refined through removing some redundant vertices and adding some missing vertices. This step will lead to a desired junction region.
(3) Finally, a handle loop along each junction region is computed as the cut of shape segmentation.

In the last step, we first use robust principal component analysis [48] for these vertices in each junction region to compute a plane. Then the intersection curve between the plane and the junction region forms the handle loop (see the blue curves surround junction regions of each model in Fig. 12). Note that a clear-cut junction and its handle loop need further processing which leads to a separate future work. The readers can refer to [45] for the details of each step of the algorithm.

Another possible application is to improve some part-aware segmentation algorithms for 3D articulated shapes [8,25] by considering the VIV function instead of some existing measures, such as the SDF [25] or VSI [8]. For instance, the SDF-based mesh

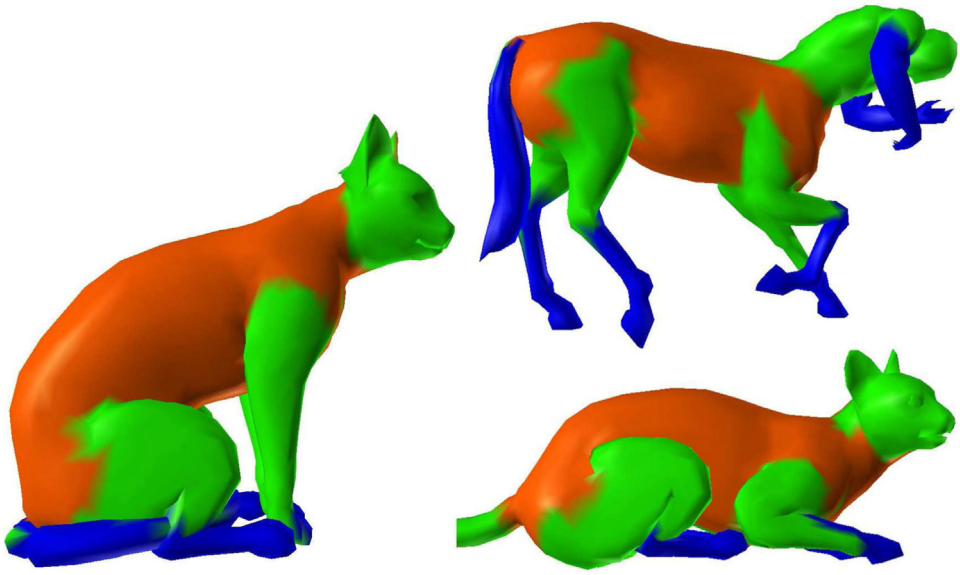


Fig. 13. Application to articulated shape segmentation, where the consistent segmentation results on mesh surfaces of several models reveal the similar parts.

segmentation algorithm in [25] is composed of two steps: (1) using soft-clustering of the mesh triangles to k clusters based on their SDF values, and (2) finding 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 [13] 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.

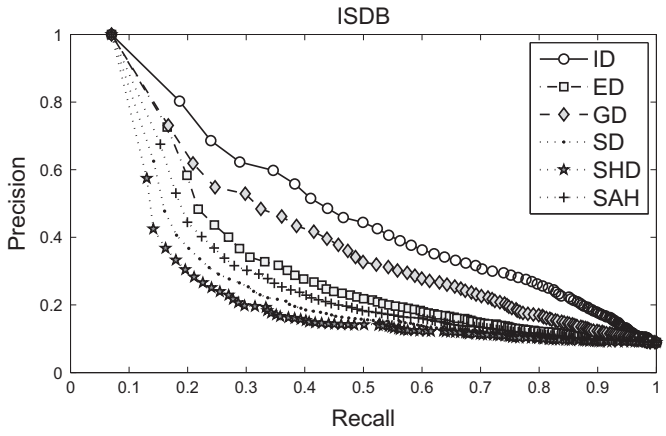


Fig. 14. Application to articulated shape retrieval. The precision–recall curves compare the retrieval results achieved with the ID descriptor versus other methods on ISDB, where the by-product visibility graph of VIV is used for computing the ID descriptor of triangular meshes.

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 [42] 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 [42,43] even for some non-closed meshes, and consequently, we can produce an approximating visibility graph between all pairs of 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 when handling small 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). For example, the VIV-based shape descriptor may fail to detect the junctions when the vertices near the junctions happen to have the similar VIV. On the other hand, it is possible that some parts are falsely detected as junctions. Fig. 15 shows an example that fails to detect the junctions, where the vertices near the horse necks (the bottom two horses) have the similar VIV. In fact, it is quite difficult to resolve this issue if just using an isolated model. However, it is of interest to improve 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 [24]. It is, however, beyond the scope of this paper, and we will leave this study to the future work.

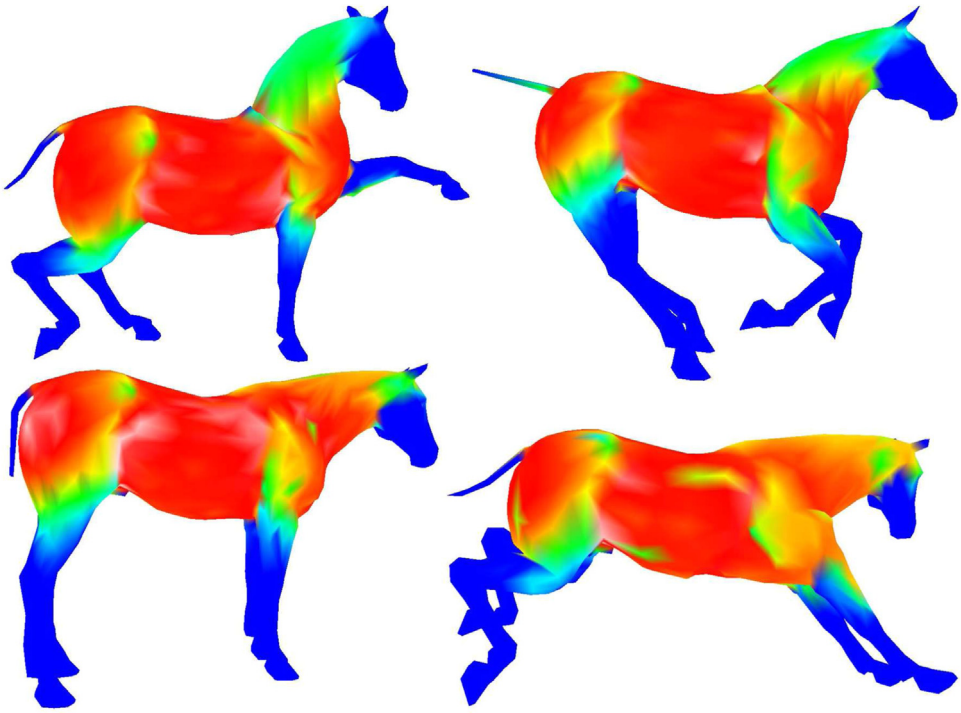


Fig. 15. Illustrating the VIV function for different poses of the same horse. Note that there is a relatively great change of VIV values on the necks of the top two horses, but there is no apparent change on the necks of the bottom two horses.

7. Conclusion

This paper considers the problem of junction-aware shape descriptor for an individual 3D model. The challenge is how to encode junction information on the boundary surface of the 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 junction detection and handle loop approximation, shape segmentation, as well as shape retrieval, which could benefit from our method.

The VIV function is a new powerful geometric measure associating volumetric information to the boundary 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 (61472202, 61272229, 61003095). The first author is also supported by the National Technological Support Program for the 12th-Five-Year Plan of China (2012BAJ03B07), and the National Key Technologies R&D Program of China (2015BAF23B03). The third author is supported by NSFC (61203317) and the Chinese 863 Program (2012AA09A409).

References

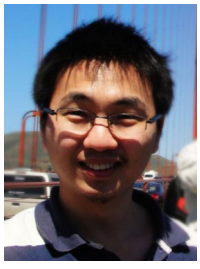
[1] A.M. Bronstein, M.M. Bronstein, A.M. Bruckstein, R. Kimmel, Analysis of two-dimensional non-rigid shapes, Int. J. Comput. Vis. 78 (1) (2008) 67–88.
[2] A.M. Bronstein, M.M. Bronstein, R. Kimmel, Numerical Geometry of Non-Rigid Shapes, Springer, New York, NY, USA, 2007.
[3] H. Ling, D. Jacobs, Shape classification using the inner-distance, IEEE Trans. Pattern Anal. Mach. Intell. 29 (2) (2007) 286–299.
[4] Y.-S. Liu, K. Ramani, M. Liu, Computing the inner distances of volumetric models for articulated shape description with a visibility graph, IEEE Trans. Pattern Anal. Mach. Intell. 23 (12) (2011) 2538–2544.
[5] C. Wang, Y.-S. Liu, M. Liu, J.-H. Yong, J.-C. Paul, Robust shape normalization of 3D articulated volumetric models, Comput.-Aided Des. 44 (12) (2012) 1253–1268.
[6] A. Ion, N.M. Artner, G. Peyré, W.G. Kropatsch, L.D. Cohen, Matching 2D and 3D articulated shapes using the eccentricity transform, Comput. Vis. Image Underst. 115 (6) (2011) 817–834.
[7] M. Shatsky, R. Nussinov, H.J. Wolfson, Flexible protein alignment and hinge detection, Proteins 48 (2) (2002) 242–256.
[8] R. Liu, H. Zhang, A. Shamir, D. Cohen-Or, A part-aware surface metric for shape analysis, Comput. Graph. Forum 28 (2) (2009) 397–406.
[9] D. Reniers, A. Telea, Part-type segmentation of articulated voxel-shapes using the junction rule, Comput. Graph. Forum 27 (7) (2008) 1845–1852.
[10] R. Su, C. Sun, T.D. Pham, Junction detection for linear structures based on Hessian, correlation and shape information, Pattern Recognit. 45 (10) (2012) 3695–3706.
[11] T. Shibuya, Fast hinge detection algorithms for flexible protein structures, IEEE/ACM Trans. Comput. Biol. Bioinform. 7 (2) (2010) 333–341.
[12] X. Chen, A. Golovinskiy, T. Funkhouser, A benchmark for 3D mesh segmentation, ACM Trans. Graph. 28 (3) (2009), Article 73.
[13] R. Gal, A. Shamir, D. Cohen-Or, Pose-oblivious shape signature, IEEE Trans. Vis. Comput. Graph. 13 (2) (2007) 261–271.
[14] Y.-S. Liu, Y. Fang, K. Ramani, Using least median of squares for structural superposition of flexible proteins, BMC Bioinform. 10 (29) (2009) 1–23.
[15] L. Parida, D. Geiger, R. Hummel, Junctions: detection, classification, and reconstruction, IEEE Trans. Pattern Anal. Mach. Intell. 20 (7) (1998) 687–698.
[16] R. Elias, R. Laganière, JUDOCA: JUnction Detection Operator based on Circumferential Anchors, IEEE Trans. Image Process. 21 (4) (2012) 2109–2118.
[17] F. Deschenes, D. Ziou, Detection of line junctions in gray-level images, in: International Conference on Pattern Recognition (ICPR'00), vol. 3, 2000, pp. 754–757.
[18] F. Zhao, P. Mendonca, R. Bhotika, J. Miller, Model-based junction detection algorithm with applications to lung nodule detection, in: ISBI'07, 2007, pp. 504–507.
[19] S.C. Flores, M.B. Gerstein, Flexoracle: predicting flexible hinges by identification of stable domains, BMC Bioinform. 8 (215), 2007.
[20] U. Emekil, D. Schneidman-Duhovny, H. Wolfson, R. Nussinov, T. Haliloglu,

Hingeprot: automated prediction of hinges in protein structures, Proteins 70 (4) (2008) 1219–1227.
[21] M. Hilaga, Y. Shinagawa, T. Kohmura, T. Kunii, Topology matching for fully automatic similarity estimation of 3D shapes, in: Proceedings of ACM SIGGRAPH, 2001, pp. 203–212.
[22] O.K.-C. Au, C.-L. Tai, H.-K. Chu, D. Cohen-Or, T.-Y. Lee, Skeleton extraction by mesh contraction, ACM Trans. Graph. 27 (3) (2008), Article 44.
[23] T. Ju, M. Baker, W. Chiu, Computing a family of skeletons of volumetric models for shape description, Comput.-Aided Des. 39 (5) (2007) 352–360.
[24] Q. Huang, V. Koltun, L. Guibas, Joint shape segmentation with linear programming, ACM Trans. Graph. 30 (6) (2011), Article 125.
[25] L. Shapira, A. Shamir, D. Cohen-Or, Consistent mesh partitioning and skeletonization using the shape diameter function, Vis. Comput. 24 (4) (2008) 249–259.
[26] L. Shapira, S. Shalom, A. Shamir, R.H. Zhang, D. Cohen-Or, Contextual part analogies in 3D objects, Int. J. Comput. Vis. 89 (2–3) (2010) 309–326.
[27] K. Hu, Y. Fang, 3D Laplacian pyramid signature, in: ACCV 2014 Workshops, Lecture Notes in Computer Science, 2015, pp. 306–321.
[28] O.K.-C. Au, Y. Zheng, M. Chen, P. Xu, C.-L. Tai, Mesh segmentation with concavity-aware fields, IEEE Trans. Vis. Comput. Graph. 18 (7) (2012) 1125–1134.
[29] C.-H. Lee, A. Varshney, D. Jacobs, Mesh saliency, ACM Trans. Graph. 24 (3) (2005) 659–666.
[30] Y.-S. Liu, M. Liu, D. Kihara, K. Ramani, Salient critical points for meshes, in: ACM Symposium on Solid and Physical Modeling (SPM'07), 2007, pp. 277–282.
[31] P. Shilane, T. Funkhouser, Distinctive regions of 3D surfaces, ACM Trans. Graph. 26 (2) (2007), Article 7.
[32] T. Funkhouser, P. Shilane, Partial matching of 3D shapes with priority-driven search, in: Symposium on Geometry Processing (SGP'06), 2006, pp. 131–142.
[33] Y. Fang, Y.-S. Liu, K. Ramani, Three dimensional shape comparison of flexible protein using the local-diameter descriptor, BMC Struct. Biol. 9 (29) (2009) 1–15.
[34] Y.-S. Liu, Y. Fang, K. Ramani, IDSS: deformation invariant signatures for molecular shape comparison, BMC Bioinform. 10 (157) (2009) 1–14.
[35] Y.-S. Liu, M. Wang, J.-C. Paul, K. Ramani, 3DMolNavi: a web-based retrieval and navigation tool for flexible molecular shape comparison, BMC Bioinform. 13 (95) (2012) 1–7.
[36] Y.-S. Liu, Q. Li, G.-Q. Zheng, K. Ramani, W. Benjamin, Using diffusion distances for flexible molecular shape comparison, BMC Bioinform. 11 (480) (2010) 1–15.
[37] K. Varanasi, Spatio-temporal modeling of dynamic 3D scenes from visual data (Ph.D. thesis), Computer Science and Applied Mathematics at the Universite de Grenoble, France, 2010.
[38] K. Varanasi, E. Boyer, Temporally coherent segmentation of 3D reconstructions, in: International Conference on 3D Data Processing, Visualization and Transmission (3DPVT), 2010.
[39] R. Gopalan, P. Turaga, R. Chellappa, Articulation-invariant representation of non-planar shapes, in: ECCV'10, 2010, pp. 286–299.
[40] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, Visibility Graphs, Computational Geometry: Algorithms and Applications, 3rd edition, Springer, New York, NY, USA, 2008 (Chapter 15).
[41] J. O'Rourke, Computational Geometry in C, 2nd edition, Cambridge University Press, New York, NY, USA, 1998.
[42] T. Ju, Robust repair of polygonal models, ACM Trans. Graph. 23 (3) (2004) 888–895.
[43] B. Adams, P. Dutré, Interactive boolean operations on surfel-bounded solids, in: Proceedings of SIGGRAPH'03, 2003, pp. 651–656.
[44] M. Desbrun, M. Meyer, P. Schröder, A. Barr, Implicit fairing of irregular meshes using diffusion and curvature flow, in: Proceedings of ACM SIGGRAPH, 1999, pp. 317–324.
[45] J. Feng, Y.-S. Liu, L. Gong, Junction-aware shape descriptor for 3D articulated models using local shape-radius variation, Signal Process. 112 (2015) 4–16.
[46] T.K. Dey, K. Li, J. Sun, D. Cohen-Steiner, Computing geometry-aware handle and tunnel loops in 3D models, ACM Trans. Graph. 27 (3) (2008), Article 45.
[47] T.K. Dey, F. Fan, Y. Wang, An efficient computation of handle and tunnel loops via Reeb graphs, ACM Trans. Graph. 32 (4) (2013), Article 32.
[48] Y.-S. Liu, K. Ramani, Robust principal axes determination for point-based shapes using least median of squares, Comput.-Aided Des. 41 (4) (2009) 293–305.

Yu-Shen Liu is an Associate Professor in the School of Software, Tsinghua University, PR China. He received his BS in mathematics from Jilin University, China, in 2000. He earned his PhD degree from the Department of Computer Science and Technology at Tsinghua University, China, in 2006. After that, he spent three years as a postdoctoral researcher at Purdue University from 2006 to 2009. His research interest is in description and retrieval of non-rigid 3D shapes.

Hongchen Deng is a Master student at Tsinghua University. His research interest is in 3D shape retrieval.

Min Liu is an Assistant Professor at the Department of Mechanical Engineering, Tsinghua University, PR China. She received her PhD degree in the School of Mechanical Engineering at Purdue University in 2008. She earned her MS in manufacturing and automation from Tsinghua University in 2001 and a BS in Mechatronics from Central South University of China in 1998. Her research interest is in geometric processing of 3D shapes.



Lianjie Gong is a Master student in the School of Software, Tsinghua University. His research interest is in 3D shape retrieval.