Pacific Graphics 2011
Bing-Yu Chen, Jan Kautz, Tong-Yee Lee, and Ming C. Lin
(Guest Editors)

Volume 30 (2011), Number 7

Computing 3D Shape Guarding and Star Decomposition

W. Yu[1] and X. Li *[2]

[1] Department of Automation, Xiamen University, China
[2] Department of Electrical and Computer Engineering, and Center for Computation and Technology,
Louisiana State University, USA. *Email: xinli@lsu.edu*

Abstract

This paper proposes an effective framework to compute the visibility guarding and star decomposition of 3D solid shapes. We propose a progressive integer linear programming algorithm to solve the guarding points that can visibility cover the entire shape; we also develop a constrained region growing scheme seeded on these guarding points to get the star decomposition. We demonstrate this guarding/decomposition framework can benefit graphics tasks such as shape interpolation and shape matching/retrieval.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Modeling—Geometric Modeling

1. Introduction

The rapid advancement of 3D scanning techniques provides massive geometric data sets nowadays with great ease. When a geometric model has very big size, its direct computation can be expensive; and when it has complex topology and geometry, directly processing the entire model can be infeasible. A common strategy to tackle these difficulties is by divide-and-conquer, which partitions the problem into solvable sub-domains. Effectively partitioning complex models can benefit many computer graphics applications relying on computationally expensive geometric processing.

This paper studies the shape partitioning based on visibility, i.e. *star decomposition*. It segments a 3D volumetric shape to a set of subregions, each of which is visible from a guarding point (such a subregion is called a star shape). It can be shown that a star-shaped subregion has some good properties. For example, harmonic volumetric parameterization can be constructed bijectively upon such domain [XHH*10]. In computer graphics and animation, star decomposition can benefit many tasks such as shape matching/retrieval, and morphing.

Surface segmentation, generally based on specific local geometric properties of surface patches, has been thoroughly examined (see survey papers [Sha08, APP*07]). Partitioning 3D objects based on their volumetric properties, such as convexity, symmetry, etc. have also been studied; however, less study has been conducted to the decomposition based on visibility. Star-decomposition is closely related to a well known *art-gallery guarding* problem. The gallery guarding problem has been studied in computational geometry community on 2D planar domains and in 2.5D for terrain guarding. But "Very little is known about gallery guarding in three dimensions" [BMKM05], especially for 3D free-form models, due to their much higher complexity.

The main contributions of this paper include:

- We develop an effective progressive integer linear programming (PILP) optimization paradigm to compute approximate optimal guarding of complex 3D free-form domains;
- We present a region-growing algorithm to compute the star-decomposition of a given 3D model, seeded from guarding points computed in the PILP;
- We explore two direct applications of our proposed guarding/decomposition framework: shape morphing and shape matching.

2. Background and Related Work

2.1. Shape Guarding

We consider the following shape guarding problem in 3D. A point $g \in M$ is *visible* to another point $p \in M$ if the line segment $\overline{gp}$ entirely locates inside M. A region (shape) M is called a *star region* (star shape) if there exists a point $g \in M$ that any point $p \in M$ is visible to g; we call such a point g a *guarding point*. All convex shapes are star shapes; but more complicated shapes usually can not be visually covered by a single guard so these regions are not star shapes. Given a general solid shape M whose boundary ∂M is a closed surface, usually tessellated by a triangle mesh, we want to find a smallest set of points $G = \{g_i\}$ inside M that every point $p \in \partial M$ is *visible* to at least one point in G.

Various versions of this problem are generally called *art gallery* problems, which are known to be a famous difficult problem. Finding minimal guards has been shown to be NP-hard for 2D polygons with holes [OS83], 2D simple polygons [LL86], and even 2D simple orthogonal polygons [SH95], using either vertex or point guards. Approximation algorithms have been studied in 1.5 ([BMKM05]) and 2D ([EHP06]) to get an close-to-optimal result in polynomial time complexity. Lien [Lie07] computes guarding for 3D point cloud data, approximating visibility using ε-view. The algorithm is based on a randomized greedy approach.

2.2. Shape Decomposition

Given a solid model M, represented by a tetrahedral mesh $\{T, V\}$, where T is the set of tetrahedra and V is the set of all the vertices, a decomposition is a partitioning of T into a set of subregions $M_i = \{T_i, V_i\}$, so that (1) $T_i \subset T$, (2) $\bigcup_i T_i = T$, and (3) $T_i \bigcap T_j = \emptyset, i \neq j$. If each subregion T_i is a star shape, we call this partitioning a star decomposition.

Shape decomposition has been widely studied in computation geometry and graphics. In computation geometry, different decomposition methods (e.g. Voronoi decomposition, convex decomposition, etc.) have been proposed; while a thorough review on other types of decompositions is beyond the scope of this work, we refer readers to surveys [CP94] and [Kei00]. In computer graphics and visualization, surface segmentation has been studied for different applications such as object recognition, meshing, skeleton extraction. Two thorough surface segmentation surveys were given in [APP*07] and [Sha08], in both of which, segmentation techniques are classified as *surface-based* methods (segmentation guided by surface properties of sub-surface-patches) and *part-based* methods (segmentation guided by volumetric properties of sub-solid-regions).

General approaches for decomposition can also be classified into two categories: *top-down* methods, by iteratively segmenting sub-parts to finer components; and *bottom-up* methods, by iteratively gluing small elements/components to larger parts. For example, the Approximate Convex Decomposition [LA06] is a top-down approach. It iteratively measures the convexity of each (sub-)region M; if it fails to satisfy the convexity criterion, we shall further *cut* it into two sub parts M_1 and M_2. The algorithm continues until all sub-regions are convex enough. A difficult issue in *top-down* methods is to find the suitable *cut* so that shapes of smaller regions become nice in a few steps. On the other hand, popular surface segmentation techniques such as region growing ([LHMR09],[CSAD04]), watershed ([MW99]), or clustering ([STK02],[GG04]) algorithms are bottom-up approaches. These approaches start from a set of seeds, then expand to include neighboring primitives (vertices, faces, tetrahedra) until their unions cover the entire region.

3. 3D Shape Guarding

Given a solid shape M, whose boundary is discretized by a triangle mesh $\partial M = \{V, F\}$, where $V = \{v_1, v_2, \ldots, v_{N_V}\}$ are vertices and $F = \{f_1, f_2 \ldots, f_{N_F}\}$ are triangle facets. Seeking fewest necessary guarding points is challenging. Our algorithm is based on the following two intuitions. (1) As demonstrated in medical visualization, medial axes (skeletons) usually have desirable visibility to the shape (referred as the "reliability" of skeletons). The skeleton can guide camera navigation and ensure full examination of the organ. (2) Hierarchical skeletons can be effectively computed, progressively reducing the size of the optimization problem and improving computation's numerical efficiency and stability against boundary perturbations or noise.

Many effective skeletonization algorithms (see a great survey [CSM07]) have been developed for 3D shapes. We use the algorithm/software of [DS06] since it efficiently generates skeletons on medial-axis surfaces of the 3D shapes. For the boundary triangle mesh ∂M with N_F triangles and the extracted skeleton with N_K nodes, the guarding problem can be converted to finding a minimal-size point set G from this N_K points, such that all N_F boundary faces are visible to G. Note that here we require each boundary triangle face is visible to G. We define the visibility of a face as follows:

- Vertex Visibility: A point $p \in M$ is *visible* to another point $q \in \partial M$ if the line segment $\overline{pq}$ connecting p and q is inside M, namely, it only intersects ∂M on q: $\overline{pq} \bigcap \partial M = \{q\}$.
- Face Visibility: A triangle face $f \in F$ is *visible* to a point p if all its three vertices are visible to p.

The shape guarding problem can be approximated as finding G to guard all the boundary vertices, in which we will only need the concept of vertex visibility. For the subsequent star decomposition purpose, to make each triangle of the boundary surface on the sub-region fully visible, we shall use the face visibility. Guarding all the faces of a region is stronger (i.e. can require more guards) than guarding all the vertices. When the triangle mesh is dense enough, face visibility well approximates the visibility in the continuous case.

3.1. Visibility Detection

A basic operation is to detect the visible region of a (skeleton) point p. Specifically, on the boundary surface $\partial M = \{F, V\}$, we define the *visibility region* $V(p)$ of an interior point p to be the collection of all visible boundary triangles: $V(p) = \{f | f \in F, f \text{ is visible to } p\}$. To compute $V(p)$, one should check intersection between each line segment $\overline{pv_i}$ and ∂M, where $v_i \in V$ is a vertex. If intersection is detected on a point $q \in \partial M$ other than v_i and the Euclidean distance $|\overline{pq}| < |\overline{pv_i}|$, then v_i is not visible from p. Simply enumerating every $\overline{pv_i}$ then detecting its intersections with every triangle $f \in F$ is time consuming: for a single skeleton point p, it costs $O(N_V \cdot N_F) = O(N_V^2)$ time to check its visibility on N_V vertices. We develop the following *sweep algorithm* to improve the efficiency.

We create a spherical coordinate system originated at p. Each vertex $v_i \in V$ is represented as $\overline{pv_i} = (r(v_i), \theta(v_i), \varphi(v_i))$, where $r(v_i) \geq 0, -\pi/2 < \theta(v_i) \leq \pi/2, -\pi < \varphi(v_i) \leq \pi$. For every triangle $f_i = (v_{i,1}, v_{i,2}, v_{i,3}) \in F, 1 \leq i \leq N_F$, its max $\theta(f_i)$ can be defined as $\theta_{max}(f_i) = \max\{\theta(v_{i,j})\}, 1 \leq j \leq 3$, the $\theta_{min}(f_i), \varphi_{max}(f_i), \varphi_{min}(f_i)$ can be defined similarly.

The segment $\overline{ov_k}$ cannot intersect with a triangle f unless

$$\begin{cases} \theta_{min}(f) \leq \theta(v_k) \leq \theta_{max}(f) \\ \varphi_{min}(f) \leq \varphi(v_k) \leq \varphi_{max}(f), \end{cases} \tag{1}$$

therefore we ignore triangles outside this range and only check ones that satisfy this condition (denoted as active triangles).

The angle functions θ and φ are not continuously defined on a sphere. When a triangle f spans $\theta = \pi$, we duplicate it to ensure that each θ of the original f is between $[\theta_{min}(f), \theta_{min}(f) + 2\pi)$ and θ of its duplicate is between $[\theta_{max}(f), \theta_{max}(f) + 2\pi)$, by adding or subtracting θ by 2π. For each triangle f spans $\varphi = \pi$, we duplicate it in the same way. Using $\theta(v_i)$ as the primary key and $\varphi(v_i)$ as the secondary key, we then sort all line segments $\overline{pv_i}$. Then we sweep all segments following the angle functions one by one in an ascending order and check intersection between the sweep line and all active triangles satisfying condition (1).

Specifically, we define a counter c_i on every triangle f_i. Initially, $c_i = 0$; when the segment $\overline{pv}, v \in f_i$ is being processed, $c_i \leftarrow c_i + 1$. The following two cases indicate that the sweep has not reached the neighborhood of the triangle f_i, and we do not need to check its intersection with line segment $\overline{pv}$:

$$\begin{array}{ll} c_i = 0 \rightarrow & \theta_{min}(f_i) > \theta(\overline{ov}), \text{ or } \varphi_{min}(f_i) > \varphi(\overline{ov}); \\ c_i > 3 \rightarrow & \theta_{max}(f_i) < \theta(\overline{ov}), \text{ or } \varphi_{max}(f_i) < \varphi(\overline{ov}). \end{array}$$

Therefore we maintain a list L of active triangles $\{f_i\}$ whose counters have $1 \leq c_i \leq 3$. When the sweep segment hits a new triangle t_j, we have $c_j = 1$ and add t_j into L; when a counter $c_j = 3$, we remove t_j from L after processing the current segment.

Given a skeleton point p, for a boundary triangle mesh with N_V vertices it takes $O(N_V \log N_V)$ to compute and sort angles of all segments. For each segment, if the size of the active triangle list L is m, it takes $O(m)$ intersection-detecting operations. Therefore, the total complexity is $O(\log N_V + mN_V)$. The incident triangles around a vertex v_i is usually very small: $m < \log N_V$. Therefore the algorithm finishes visibility detection of p in $O(N_V \log N_V)$ time. On a skeleton containing N_K nodes, it takes $O(N_K N_V \log N_V)$ pre-computation time to know the visibility region for all nodes.

3.2. Greedy and Optimal Guarding

Once visibility regions for all skeletal nodes are computed, the guarding problem can be converted into a set-covering problem. Consider a set in which each element corresponds to a face on the boundary; a skeleton node can see many faces so it covers a subset of elements. We want to pick several skeleton nodes so that the union of their covered subsets is the entire set. The set-covering problem, shown to be NP-complete [KKK83], can be formally defined as follows: given the universe $V = \{v_i\}$ and a family S of subsets $S_j = \{s_{j,k}\}, s_{j,k} \in V$, a cover is a subfamily $C \subset S$ of sets whose union is V. We want to find a covering C that uses the fewest subsets in S. C indicates an optimal subset of skeletal nodes that can guard the entire region. Skeletons generated using medial-axis based methods with dense enough nodes usually ensure S itself is a covering. This holds in all our experiments. If a coarsely sampled skeleton can not cover the entire V, we further include all those invisible vertices into the skeleton point set.

A **greedy** strategy for the set covering is as follows: iteratively pick the skeletal nodes that can cover the most faces in V, then remove all guarded faces from V (and update S accordingly since the universe becomes smaller), until $V = \emptyset$. Such a greedy strategy is quite effective and it yields $O(\log n)$ approximation [Joh73] to the set covering problem.

An **optimal** selection can be computed by $0-1$ programming, also called Integer Linear Programming (ILP). We assign a variable x_i on each skeleton node p_i: $x_i = 1$ if p_i is chosen, and $x_i = 0$ otherwise. The *objective function* to minimize is then $\sum_{i=1}^{m} x_i$.

Every element should be visible, for $\forall f_i \in F$ visible to some skeletal nodes $P_i = \{p_{(i,1)}, \ldots, p_{(i,k)}\}$, at least one node in P_i should be chosen to ensure f_i guarded. Thus we solve:

$$\min \sum_{i=1}^{m} x_i, \text{ subject to} \tag{2}$$

$$x_i = 0, 1, \text{ and } \sum_{j \in J(i)} x_j \geq 1, \forall i \in \{1, \ldots, n\}, \tag{3}$$

where $J(i)$ is the index set of nodes p_j visible to f_i.

The above optimization can be solved using branch-and-bound algorithms. When the dimension is small (e.g. a few hundreds to a few thousands), we can use the TomLab Optimization package [Hol98] to solve it efficiently.

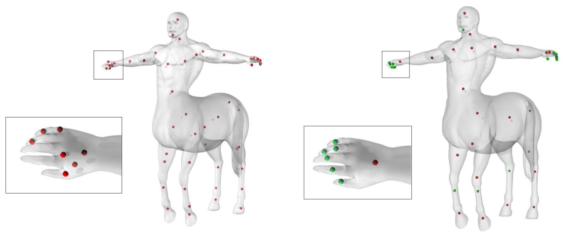

Figure 1: *Greedy (left) and PILP (right) Guarding of Centaur.*

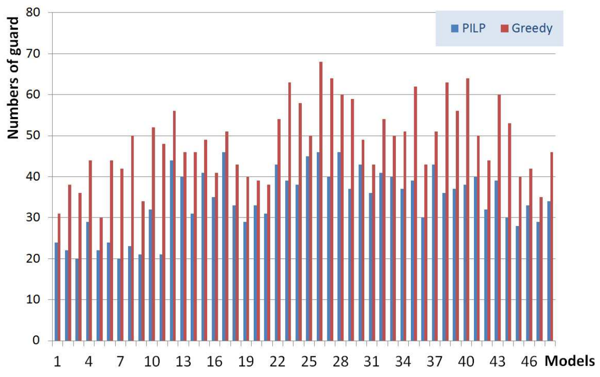

Figure 2: *Greedy vs PILP on 48 models in TOSCA dataset [TOS]. The x-axis lists the 48 models, the y axis indicates the necessary guards computed. The blue bar indicates the PILP result and red is the greedy result. PILP has similar computational performance with the greedy approach, but generate better guarding (on average, using 20% less guards).*

3.3. Progressive Guarding

Directly solving optimal guarding is expensive, or prohibitive for big models. In contrast, greedy algorithm generates many unnecessary guards and is sensitive to local boundary geometric perturbations. Therefore, we propose a progressive integer linear programming (PILP) framework using progressive mesh simplification and refinement [Hop96], adaptively combining ILP and multiresolutional refinement.

We progressively simplify the boundary mesh ∂M into several resolutions $\partial M^i = \{T^i, V^i\}, i = 0, \ldots, m$. On each level we keep the problem size within the scale that ILP can solve. The finest skeleton is re-sampled with joints preserved and used for computation. In the coarsest level $i = 0$, we solve the optimal guarding using ILP. Then we progressively move to finer levels with more details. On each finer level i, we greedily remove regions covered by existing guards computed in level $i-1$, then again use ILP to find necessary new guards. With we move toward finer level with increased details, new guards are added until the finest resolution ∂M^m is covered. On every new level, we also resolve a few least significant existing guards (whose visibility region covers a small area $Area(V(p)) < \varepsilon Area(M^i)$). We do not directly insert them into the current level's guarding set G^i and do not remove their covered boundary faces from the universe.

On each level, we also conduct four **reduction** operations before ILP computation. These reductions significantly reduce the optimization problem size. Suppose we store the visibility information in an incidence matrix A: $a_{ij} = 1$ if the skeletal node p_i can see the face f_j, and $a_{ij} = 0$ otherwise. Originally the A is $|N_K| \times |N_F|$, and we apply the following **four rules** to reduce the size of A:

1. If column j has only one non-zero element at row i, we must pick p_i in order to see v_j. Therefore, add p_i into G, remove column j. Also, for all non-zero element a_{ik}, remove column k (since all vertices visible to p_i are now covered, and can be removed).
2. If row i_1 has all its non-zero elements non-zero in row i_2, i.e. $a_{i_1,j} = 1 \rightarrow a_{i_2,j} = 1$, then p_{i_2} sees all vertices that p_{i_1} can see, and we can remove the entire row i_1.
3. If column j_1 has all its non-zero elements non-zero in column j_2, i.e. $a_{i,j_1} = 1 \rightarrow a_{i,j_2} = 1$, then guarding v_{j_1} guarantees the guarding of v_{j_2}, and we can remove the entire column j_2.
4. If the matrix A is composed of several blocks, we partition A to several small matrixes $\{A_k\}$.

In step 4, since we remove faces that have been covered by existing guards, remaining boundary faces could be partitioned to several connected-components far away from each other. And these sub-components may be optimized separately, which significantly reduces optimization size.

This PILP scheme can efficiently compute the guarding for large size 3D volumetric regions and generate a hierarchical guarding graph. The pipeline is fully automatic, and furthermore, it has the following important **advantages** over both the pure greedy strategy and global ILP optimization (more statistical results are shown in Table 1 and Fig. 2). Note that the t_P in Table 1 does not include the progressive mesh computation time. However, progressive mesh can be computed efficiently. Simplifying a 10k mesh takes roughly 10 seconds. From our experiments, we can see that

- PILP is **much faster** than global ILP. Its computational efficiency is improved for several orders of magnitude over ILP on large-size geometric models, and can therefore handle massive data.
- With comparable speed to the greedy approach, PILP usually provides **much better** guarding solutions. Firstly, the PILP guards number is smaller than the greedy approach; secondly, the PILP guarding is hierarchical and therefore is **robust** against geometric noise (Figure 1 shows an example. In PILP, global structure from coarser levels is stable under local refinement to new details).

In our experiments, we simplify the boundary mesh to the coarsest level with 5k vertices for the first round ILP optimization. On each iteration, we refine to next level with additional 10k vertices. When the size of constraints is around 5k, and the size of variables (skeletal nodes) is around 1k,

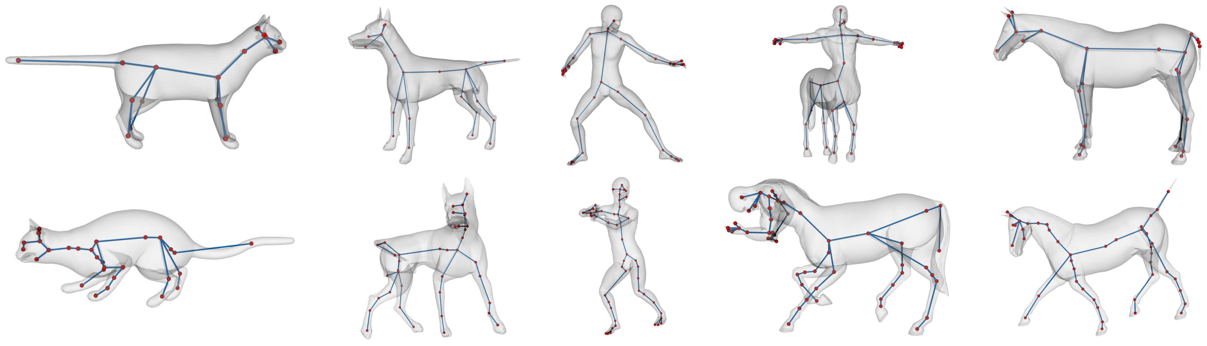

Figure 3: *Visualizing Guarding of Models in TOSCA dataset.*

Table 1: *Guarding Statistics. N_V is the boundary surface vertex number. N_I, N_G, N_P indicate the number of necessary guards computed by ILP, Greedy, and PILP approaches, respectively. t shows the computational time in seconds. Guarding of big models cannot be solved directly using ILP, so their statistics are not applicable.*

Models (N_V)	N_I	N_G	N_P	t_I	t_G	t_P
Greek (9,994)	15	22	18	4,122.4	290.2	293.1
David (9,996)	16	22	17	107,391.2	233.9	235.2
Female (10,002)	13	18	14	2,046.2	264.2	281.1
Male (10,002)	14	16	15	3,074.3	298.6	310.2
Cat (10,004)	14	19	15	3173	375.4	393.1
Wolf (10,005)	13	18	15	8044	328.1	349.9
Dog (15,002)	–	39	27	–	412.3	433.2
Victoria (15,000)	–	35	27	–	408.7	421.2
Horse (20,002)	–	38	29	–	376.1	384.2
Michael (20,002)	–	46	31	–	321.0	332.9
Gorilla (30,004)	–	60	46	–	462.4	490.1
Centaur (30,002)	–	52	32	–	488.1	514.5

the optimization takes 10-30 seconds to solve. We set the significance threshold parameter to be $\varepsilon = 10\%$.

Figures 1 shows an example of guarding the Centaur model, where we can see the PILP guarding provides a stable hierarchial guarding. The guards added in the finest level are colorized in green while the one computed on coarser level are rendered in red. We perform extensive experiments on our new algorithm. And it demonstrates great effectiveness. More guarding results are visualized in Figure 3. Statistical comparison is shown in Table 1; a more thorough comparison chart between greedy and PILP approaches on 48 models from the TOSCA dataset is depicted in Figure 2. As we can see in this table and the chart, PILP has similar optimality as the ILP solution, but is much faster; while compared with greedy approach, PILP has similar efficiency, but provides the guarding 20% better than that of greedy method on average. Considering that the greedy approach is generally a nice approximation for this problem, the guarding generated by PILP is very nice.

Discussion. It should be noted that theoretically our algorithm solves an approximate optimal solution. In our current computational framework, two aspects need to be considered on the approximation of the optimal guarding problem. (1) We enforce the *face visibility* of the guarding. This can be considered as an approximation to guarding all points on the boundary. However, when the boundary surface mesh is dense enough with respect to the geometry of the model, we usually can assume this guarding is accurate enough because it is unlikely to have a branch that blocks the interior region of a face while leaving its all three vertices visible. (2) We compute the guarding points from the curve skeleton. Because of this, our simplified problem setting is not guaranteed to get the optimal solution. We perform experiments to evaluate whether using guarding candidates from the skeleton leads to larger guarding point set. Table 2 shows results of these experiments: guarding using skeletal candidates produces the same optimality compared with guarding computed using all tetrahedral vertices. Therefore, our intuition of choosing guards from skeleton is experimentally justified; our approximation is close to the optimal solution.

Table 2: *Skeletal-Nodes Guarding versus Tetrahedral-Vertices Guarding. $|V_{\partial M}|$ and $|V_M|$ are numbers of boundary vertices and tetrahedral vertices, respectively; $|S|$ is the number of nodes on extracted skeletons; $|G_{V_M}|$ and $|G_S|$ are the sizes of computed guarding sets (solved by ILP) when using all tetrahedral vertices as candidates and using only skeletal nodes as candidates, respectively.*

Models	$\lvert V_{\partial M}\rvert$	$\lvert V_M\rvert$	$\lvert S\rvert$	$\lvert G_{V_M}\rvert$	$\lvert G_S\rvert$
Kitten	400	1,682	122	3	3
Beethoven	502	2,895	88	2	2
Bimba	752	5,115	139	2	2
Buddha	502	3,002	155	2	2
Bunny	998	8,320	270	5	5

4. Star Decomposition

Guarding points are natural seeds to start region growing for the star decomposition. The sweep algorithm (Section 3.1) can be generalized to tetrahedral mesh vertices so that visibility among vertices and guards can be efficiently precomputed. We start region growing from all guards while simultaneously preserving star-property on all subregions.

Given the tetrahedral mesh $M = \{T, V\}$ where T and V are the sets of tetrahedra and vertices, we start from the guarding points $G = \{g_i\}, i = 1, \ldots, K_g$, and assign a specific color-value c_i on each guard g_i. The growing procedure can then be illustrated as assigning a unique color c_i to each tetrahedron, so that at the end, the connected component in color c_i is a star shape guarded by g_i. We can grow sub-regions on the dual graph of the tetrahedral mesh using the following notations and operations.

Similar to the face visibility, we say that a tetrahedron is visible from a point g if all its four vertices are visible from g. Given a guarding point g and a tetrahedron $t \in T$, we define the *visibly dependent* tetrahedral set $T_v(g,t) = \{t_i\}$ such that it contains all the tetrahedra that one of the four ray segments $\overline{gv_j}, v_j \in t$ passes through. In other words, t is visible to g if $\forall t \in T_v(g,t)$ is visible to g.

We then denote the *one-ring faces* surrounding a vertex v as $F_n(v)$: $F_n(v) = \{f | \forall t \in T, v \subset t, f \subset t, v \not\subset f\}$; and denote the neighboring tetrahedra of t as $T_n(t) = \{t' | t' \in T, \forall v \subset t, F_n(v) \subset t'\}$; then we can define $\widetilde{T}_v(g,t) = T_v(g,t) \cap T_n(t)$. Intuitively, including tetrahedra in $\widetilde{T}_v(g,t)$ into a sub-region guarded by g prevents triangles in $F_n(v_j)$ from becoming the boundary of this sub-region (which could block the visibility of t from g). It is not difficult to further show that

a) t is visible if $\forall t' \in \widetilde{T}_v(g,t)$ are visible.
b) t can be safely added into a sub-region M_g seeded in g without violating its star-property, if all $t' \in \widetilde{T}_v(g,t)$ are in M_g.

The dual graph D of the given tetrahedral mesh is defined in the following way: a **node** $n_i \in D$ is defined for each tetrahedron t_i. For a node n_i visible from g_k (with the color c_k), we create a **directed edge in color** c_k to n_i from another node n_j if $t_j \in \widetilde{T}_v(g_k, t_i)$; and we call n_j is a **color-**c_k **predecessor** of n_i. Since recursively, $T_v(g_k,t_i) = \widetilde{T}_v(g_k,t_i) \cup T_v(g_k,t_j)$, we only need to store each node's visibility dependency relationship. $\widetilde{T}_v(g_k,t_i)$ can be computed in $O(1)$ time by checking the intersections of $\overline{g_k, v}$ and $F_n(v)$ for each $v \in t_i$.

For each guard g_k we generate a virtual node in D and connect it to nodes corresponding to all its one-ring tetrahedra. Then each guard g_k and its visible region defines a direct acyclic graph R_k. The entire 3D region guarded by K_g points $\{g_i\}$ corresponds to a connected graph with K_g sources. Each source g_i has an individual color c_i, the region growing assigns each node a unique color. A node n_j can be assigned by a color c only when all its color-c predecessors (on which n_j is visible dependent) are already assigned by color-c.

The region growing on the dual graph can be applied using the **node-merging**. When there is a color-c edge from a color-c node n_i to an uncolored node n_j, and all edges *entering* n_j are leaving from color-c nodes, then n_j can be safely colored by c. Therefore we can merge them together to one node in color-c, preserving all distinct outgoing edges. The region growing procedure is converted to iteratively merging each uncolored node to one of the K_g "growing" colored nodes. Therefore, the region growing algorithm can be summarized as follows.

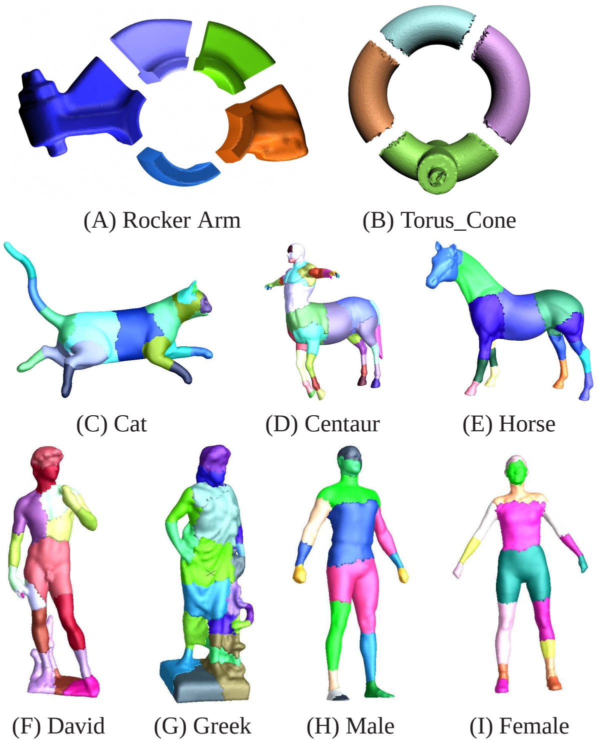

Figure 4: *Star Decomposition of Solid Models. Different subparts are rendered in different colors.*

Cost Function. For each node n_i, we can compute how many nodes are directly or indirectly visually dependent on n_i with respect to g_k. This can be pre-computed in linear time: after the dependency graph R_k is created from the source g_k to leafs, inversely the dependency cost can be accumulated and stored as $f(g_k, n_i)$.

Nodes Merging. We merge uncolored nodes with colored region based on the cost function $f(g,n)$. A node shall merge into a growing region that can see it and has biggest corresponding cost. This repeats until all nodes are colorized or no node can be further merged. If a node cannot be given the color of its any entering edges, it is left unclassified after the region growing. We collect each uncolored connected components, and respectively compute their guarding and re-apply the region growing until all tetrahedra are colored.

Computational Complexity. For each guard, the preprocessing step computes the visibility dependency in $O(m_V \log m_V)$, where m_V is the number of tetrahedral vertices. Computing the visibility dependency of one tetrahedral vertex v takes $O(K_n)$, where K_n is v's one-ring tetrahedra. Since $K_n < \log m_V$, the preprocessing time for each guard is $O(m_V \log m_V)$. The region growing can be finished in $O(m_T)$, where m_T is the number of tetrahedra. So the total

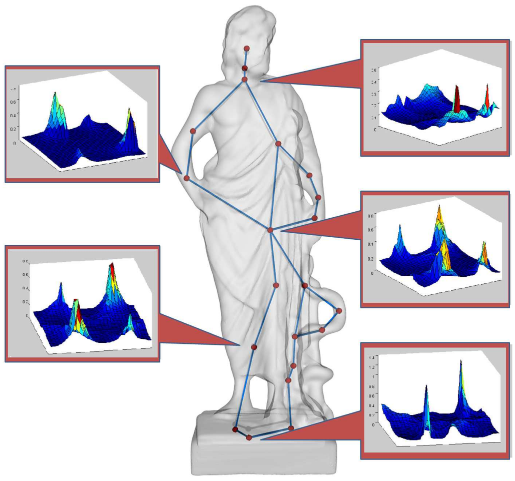

Figure 5: *Shape Descriptor of Greek Sculpture. Each histogram stores distances from each guarding point to the boundary surface along sampling directions.*

decomposition complexity is $O(K_g m_V \log m_V + m_T)$, where K_g is the number of guards. For example, it takes about 550 seconds to perform the star decomposition on solid David (175,079 tetrahedra, 17 guards), including 400 seconds in guarding computation. Figure 4 illustrate our decomposition results on many solid models.

The visibility dependency relationship is the sufficient condition to guarantee the tetrahedron visibility. A tetrahedral sub-region that grows following this dependency relationship is guaranteed to be star-shaped. However, this constraint is stronger than necessary, especially when tetrahedral mesh is sparse, and some tetrahedra near inner partitioning boundary may not be considered acceptable during the region growing. In practice, in order to include these tetrahedra, we release this constraint by accepting a tetrahedron if two to three of its vertices are visible. Finally, we perform a Laplacian smoothing step on the inner-border of sub-regions after the region growing. This further moves these interior tetrahedral vertices, and the smoothed boundary improves the decomposition result: more than 99% tetrahedral vertices are visible from their corresponding guards.

5. Applications

Our proposed guarding and decomposition framework can benefit many geometric processing tasks. In this section, we demonstrate two direct applications in computer graphics: shape matching and shape morphing.

5.1. Shape Matching and Retrieval

We define a descriptor for a shape based on its guarding. The descriptor has two parts: the guarding skeleton (or guarding graph) $\mathcal{G}$ and histograms $\mathcal{H}$ defined on nodes.

The guarding graph $\mathcal{G}$ is a graph extracted following the skeletal graph, whose nodes are the guarding point set $G = \{g_i\}$. At each guard g_i, we compute a histogram $\mathcal{H}(g_i)$ storing the distances from g_i to the object boundary surface towards a set of sampling directions. Specifically, the histogram is constructed as follows. From each guarding point p, we shoot rays $\{r_i\}$ towards all spatial directions defined on a unit sphere. Each ray r_i intersects with ∂M on a point q_i. The length of the line segment $\overline{pq_i}$ is stored in the histogram H_p. This histogram captures the geometry near this point. Fig. 5 illustrates the descriptor of the Greek sculpture.

The proposed descriptor has two good properties:

- *Completeness*. The geometry of the original shape can be completely reconstructed from its descriptor. Distance-to-boundary distributions nicely capture geometry characteristic of the solid shape and are suitable for the matching purpose. The guarding graph can visibly covers the entire region, so the shape descriptor is complete.
- *Conciseness*. The graph structure $\mathcal{G}$ has fewest necessary nodes because the guarding is optimized. Therefore, descriptor matching is efficient.

5.1.1. Matching Shape Descriptors

To compare two 3D models M and M', we match their guarding graphs $\mathcal{G} = \{G, E_{\mathcal{G}}\}$ and $\mathcal{G}' = \{G', E_{\mathcal{G}'}\}$, where vertex sets G, G' are guarding point set and edge sets $E_{\mathcal{G}}, E'_{\mathcal{G}}$ following the adjacency of the decomposition. We match them by solving a deformation of one skeleton to fit the other. The deformation is guided by a weighted energy $E_{\mathcal{G}}(\mathcal{G}, \mathcal{G}')$ composed of three terms: (1) the matching error E_M on each node, (2) the smoothness error E_S on the deviation of the transformations of two adjacent nodes, and (3) the length-preserving error E_L on each edge. Formally, suppose we define the affine transformation ϕ_i on each node g_i of the guarding graph $\mathcal{G}$, then these three terms are:

$$\begin{aligned} E_M &= \textstyle\sum_{g_i \in G} D(\phi_i(g_i), G')^2, \\ E_S &= \textstyle\sum_{[g_i,g_j] \in E_{\mathcal{G}}} ||\phi_i - \phi_j||_2, \\ E_L &= \textstyle\sum_{[g_i,g_j] \in E_{\mathcal{G}}} (||\phi_i(g_i) - \phi_j(g_j)|| - ||g_i - g_j||)^2, \end{aligned} \tag{4}$$

where $D(\phi_i(g_i), G')$ denotes the distance from the transformed point $\phi_i(g_i)$ to the guarding skeleton. In practice, we integrate two costs: (1) geometric distance, approximated using the distance from $\phi_i(g_i)$ to its closest point in G', and (2) topology distance, represented by the valence information of g_i. Therefore each node is represented as a 4-dimensional vector and the distance between two nodes is computed using the L_2 norm. During the optimization, this shortest distance can be efficiently recomputed using a k-d tree data structure. In E_S, the $||\phi_i - \phi_j||_2$ is the L_2 norm of the transformation matrix $\phi_i - \phi_j$. In E_L, the $||g_i - g_j||, ||\phi_i(g_i) - \phi_j(g_j)||$ denote the distance between adjacent points before and after the transformation.

The final objective function $E_{\mathcal{G}}(\mathcal{G}, \mathcal{G}')$ is a quadratic weighted sum of these cost functions:

$$E_{\mathcal{G}}(\mathcal{G}, \mathcal{G}') = \alpha_1 E_M + \alpha_2 E_S + \alpha_3 E_L, \tag{5}$$

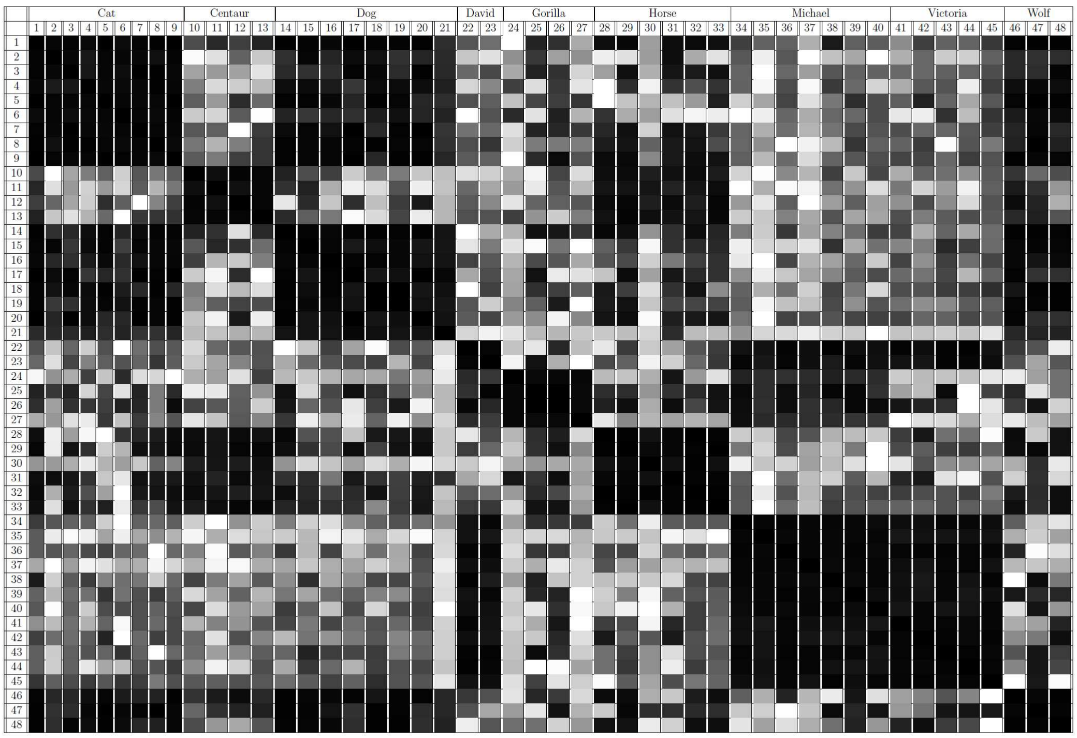

Figure 6: *Shape retrieval experiment conducted in the TOSCA dataset of 48 models. Black indicates better similarity. Those blocks of black regions indicate the following groups are more similar:* $\{$*cats, dogs, wolves*$\}$, $\{$*David,Michael,Victoria*$\}$, $\{$*horses,centaurs*$\}$, *and ect.*

where in our experiments we set $\alpha_1 = 0.1, \alpha_2 = 1, \alpha_3 = 1$. We compute transformations defined on all the nodes by minimizing $E_{\mathcal{G}}(\mathcal{G}, \mathcal{G}')$; the solution is a non-rigid mapping between $\mathcal{G}$ and $\mathcal{G}'$. The quadratic optimization can be solved efficiently.

After the transformation is computed, we add in the histogram matching error

$$E_H = \sum_{g_i \in G} ||\mathcal{H}(g_i) - \mathcal{H}(g'_j)||_2,$$

where g'_j denotes the closest point in G' under the previous matching. The difference between two histograms is again measured using the L_2 norm. The shape matching energy is

$$E(M, M') = E_{\mathcal{G}} + \alpha_4 E_H, \tag{6}$$

where we set $\alpha_4 = 0.5$.

Finally, we use the symmetric energy $(E(M,M') + E(M',M))/2$ as the shape distance between M and M'.

5.1.2. Shape Retrieval

For all the shapes in the database we can pre-compute their guarding graphs and node histograms as their descriptors. Given a new object we compute its descriptor, then match it with existing descriptors following the above approach.

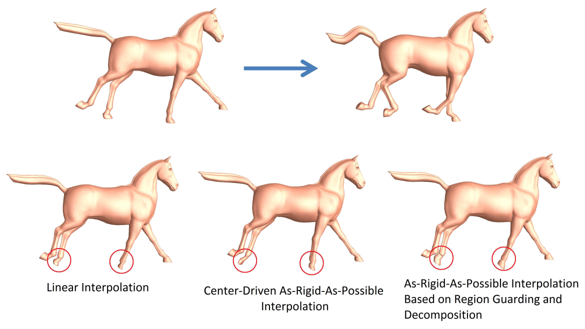
Linear Interpolation
Center-Driven As-Rigid-As-Possible Interpolation
As-Rigid-As-Possible Interpolation Based on Region Guarding and Decomposition

Figure 7: *Shape Morphing. The top row shows the source and target shapes. The lower row shows the* 50% *shape interpolation.*

The descriptor with the smallest matching error indicates the most similar object in the database. We perform shape matching and comparison on the TOSCA dataset. The comparison results are illustrated in Fig. 6, where black indicates small difference. The black blocks in this figure indicates several groups of models, although in different postures, share better similarity. For examples, cats$-$dogs$-$wolves, David$-$Michael$-$Victoria, horses$-$centaurs, and ect.

5.2. Shape Morphing

Given the source surface M_1 and target M_2, we want to compute an interpolation $M(t), 0 \leq t \leq 1, M(0) = M_1, M(1) = M_2$. Interpolation between M_1 and M_2 can be generated through the *consistent guarding*. Consistent guarding of M_1 and M_2 are two isomorphic graphs G_1 and G_2, such that G_1 (G_2) guards M_1 (M_2) respectively. The consistent guarding $\{G_1, G_2\}$ can be computed in three steps:

1) Compute cross-surface parameterization $f_M : M_1 \rightarrow M_2$ using surface mapping techniques (e.g. [LBG*08][LGQ09]);
2) Extracting compatible skeletons (e.g. [ZST*10]) that bijectively corresponds the first curve skeleton C_1 (of M_1) to the second skeleton C_2 (of M_2), $f_C : C_1 \rightarrow C_2$;
3) Solve PILP simultaneously. We say $v_i \in M_1$ and $f_M(v_i) \in M_2$ are *simultaneously visible* to $p_j \in C_1$ and $f_C(p_j) \in C_2$, if both v_i is visible to p_j and $f_1(v_i)$ is visible to $f_2(p_j)$.

The solution found in Step-3 is two consistent guarding sets $\{G_1, G_2\}$. G_i may contain more guards than necessary to cover M_i, but the guarding points and their images consistently cover both models. This consistent guarding can generate consistent star decomposition which can benefit many applications. An example is shape interpolation.

Conventionally, shape morphing can be generated by linear interpolation: given inter-surface mapping $f_M : M_1 \rightarrow M_2$, the morphing for each vertex is generated by linear interpolation between $v_1 \in M_1$ and its image $f_M(v_1) \in M_2$: $v(t) = (1-t)v_1 + tv_2$.

With star decomposition, we can interpolate the corresponding regions to generate the morphing. A similar idea in 2D, based on star decomposition for 2D polygons and the interpolation of polar coordinates, is introduced in [SR95]. However, directly generalizing this to 3D by interpolating the spherical coordinates does not work well. We break the interpolation into the rigid part and the non-rigid part.

Rigid part. After the consistent star decomposition, we get the consistent surface segmentation $\{S_1, S_2\}$, for each subregion with m triangles $\{P_1, P_2, P_3, \ldots, P_m\}$ in S_1, there is a corresponding triangle set in S_2, $\{Q_1, Q_2, Q_3, \ldots, Q_m\}$, their guards are $g_{S_1} = (g^x_{S_1}, g^y_{S_1}, g^z_{S_1})$ and $g_{S_2} = (g^x_{S_2}, g^y_{S_2}, g^z_{S_2})$ respectively, for each corresponding triangle pair P, Q, we compute the Jacobian of the affine transformation A_T:

$$A_T = \begin{bmatrix} p^x_1 - g^x_{S_1} & p^y_1 - g^y_{S_1} & p^z_1 - g^z_{S_1} \\ p^x_2 - g^x_{S_1} & p^y_2 - g^y_{S_1} & p^z_2 - g^z_{S_1} \\ p^x_3 - g^x_{S_1} & p^y_3 - g^y_{S_1} & p^z_3 - g^z_{S_1} \end{bmatrix}^{-1} \cdot \begin{bmatrix} q^x_1 - g^x_{S_2} & q^y_1 - g^y_{S_2} & q^z_1 - g^z_{S_2} \\ q^x_2 - g^x_{S_2} & q^y_2 - g^y_{S_2} & q^z_2 - g^z_{S_2} \\ q^x_3 - g^x_{S_2} & q^y_3 - g^y_{S_2} & q^z_3 - g^z_{S_2} \end{bmatrix}$$

where $P = \{p_1, p_2, p_3\}$, $p_i = (p^x_i, p^y_i, p^z_i), i = 1,2,3$ is the the coordinates of the ith vertex of triangle P. Given a $t, 0 < t < 1$, we interpolate the Jacobian by polar decomposition [ACOL00]. Since a vertex may be shared by several triangle pairs, each triangle pair has a transformation, to keep the mesh consistent during the interpolation, we compute the interpolation vertex i position $v^i_r(t)$ by minimizing the quadratic error between the actual Jacobian and the desired ones, as stated in [BBA08].

Non-rigid part. Excluding the rigid part transformation, we use a linear interpolation to blend the non-rigid deformation. For vertex i we compute the $v^i_n(1) = v^i_{M_2} - v^i_r(1)$, where $v^i_{M_2}$ is the target position, $v^i_r(1)$ is the rigid transformed position. Then we compute the non-rigid position $v^i_n(t) = tv^i_n(1)$. So the final interpolated position of vertex i is $v^i(t) = v^i_r(t) + v^i_n(t)$.

Compared with linear interpolation and the as-rigid-as-possible interpolation [BBA08] directly computed globally, our morphing based on star-decomposition could lead to less self-intersection and therefore generate more natural interpolation: A comparative example is shown in Figure 7. The source and target models are shown in the first row. In the second row, from left to right, the 50% morphing generated by linear interpolation, global center-driven as-rigid-as-possible interpolation, and our as-rigid-as-possible interpolation based on star decomposition are illustrated. Our result is natural, especially can be seen at regions in red circles.

6. Conclusion and Future Work

In this paper we present an efficient progressive integer linear programming scheme to compute 3D shape guarding and star-decomposition. The proposed method is efficient and robust, which is demonstrated by extensive experiments. We also explore its effective computer graphics applications in shape retrieval and shape morphing.

Skeleton shape and skeletal nodes sampling are important for our guarding compaction. We will develop greedy or optimization strategies to further adjust them during the guarding computation. We will also improve our shape matching algorithm, and explore new applications of guarding and star decomposition.

7. Acknowledgements

This work is partially supported by Louisiana Board of Regents Research Competitiveness Subprogram (RCS) LEQSF(2009-12)-RD-A-06, LA Board of Regents PFund:NSF(2011)-PFund-236, and LSU Faculty Research Grant 2010. Part of the work was done when the first author was a visiting student at Center for Computation and Technology, LSU.

References

[ACOL00] ALEXA M., COHEN-OR D., LEVIN D.: As-rigid-as-possible shape interpolation. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), SIGGRAPH '00, pp. 157–164. 9

[APP*07] AGATHOS A., PRATIKAKIS I., PERANTONIS S., SAPIDIS N., AZARIADIS P.: 3d mesh segmentation methodologies for cad applications. *Computer-Aided Design 4*, 6 (2007), 827–841. 1, 2

[BBA08] BAXTER W., BARLA P., ANJYO K.: Rigid shape interpolation using normal equations. In *Proc. Non-Photorealistic Animation and Rendering* (2008). 9

[BMKM05] BEN-MOSHE B., KATZ M. J., MITCHELL J. S. B.: A constant-factor approximation algorithm for optimal terrain guarding. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms* (2005), pp. 515–524. 1, 2

[CP94] CHAZELLE B., PALIOS L.: Decomposition algorithms in geometry. *Algebraic Geometry and Its Applications* (1994), 419–447. 2

[CSAD04] COHEN-STEINER D., ALLIEZ P., DESBRUN M.: Variational shape approximation. In *ACM SIGGRAPH* (2004), pp. 905–914. 2

[CSM07] CORNEA N., SILVER D., MIN P.: Curve-skeleton properties, applications, and algorithms. *Visualization and Computer Graphics, IEEE Transactions on 13*, 3 (may. 2007), 530 –548. 2

[DS06] DEY T., SUN J.: Defining and computing curve-skeletons with medial geodesic function. In *Proc. Eurographics Symp. on Geometry Processing* (2006), pp. 143–152. 2

[EHP06] EFRAT A., HAR-PELED S.: Guarding galleries and terrains. *Inf. Process. Lett. 100*, 6 (2006), 238–245. 2

[GG04] GELFAND N., GUIBAS L. J.: Shape segmentation using local slippage analysis. In *Proc. Symposium on Geometry processing* (2004), pp. 214–223. 2

[Hol98] HOLMSTROM K.: Tomlab – a general purpose, open matlab environment for research and teaching in optimization, 1998. 3

[Hop96] HOPPE H.: Progressive meshes. In *SIGGRAPH* (1996), pp. 99–108. 4

[Joh73] JOHNSON D. S.: Approximation algorithms for combinatorial problems. In *STOC '73: Proceedings of the fifth annual ACM symposium on Theory of computing* (New York, NY, USA, 1973), ACM, pp. 38–49. 3

[Kei00] KEIL J. M.: Polygon decomposition. *Handbook of Computational Geometry* (2000). 2

[KKK83] KAHN J., KLAWE M., KLEITMAN D.: Traditional galleries require fewer watchmen. *SIAM Journal on Algebraic and Discrete Methods 4*, 2 (1983), 194–206. 3

[LA06] LIEN J.-M., AMATO N. M.: Approximate convex decomposition of polygons. *Computational Geometry 35*, 1-2 (2006), 100–123. Special Issue on the 20th ACM Symposium on Computational Geometry. 2

[LBG*08] LI X., BAO Y., GUO X., JIN M., GU X., QIN H.: Globally optimal surface mapping for surfaces with arbitrary topology. *IEEE Trans. on Visualization and Computer Graphics 14*, 4 (2008), 805–819. 9

[LGQ09] LI X., GU X., QIN H.: Surface mapping using consistent pants decomposition. *IEEE Transactions on Visualization and Computer Graphics 15*, 4 (2009), 558–571. 9

[LHMR09] LAI Y.-K., HU S.-M., MARTIN R. R., ROSIN P. L.: Rapid and effective segmentation of 3d models using random walks. *Comput. Aided Geom. Des. 26*, 6 (2009), 665–679. 2

[Lie07] LIEN J.-M.: Approximate star-shaped decomposition of point set data. In *Eurographics Symposium on Point-Based Graphics* (2007). 2

[LL86] LEE D. T., LIN A. K.: Computational complexity of art gallery problems. *IEEE Trans. Inf. Theor. 32*, 2 (1986), 276–282. 2

[MW99] MANGAN A. P., WHITAKER R. T.: Partitioning 3D surface meshes using watershed segmentation. *IEEE Transactions on Visualization and Computer Graphics 5*, 4 (1999), 308–321. 2

[OS83] O'ROURKE J., SUPOWIT K.: Some np-hard polygon decomposition problems. *Information Theory, IEEE Transactions on 29*, 2 (mar 1983), 181 – 190. 2

[SH95] SCHUCHARDT D., HECKER H.-D.: Two np-hard art-gallery problems for ortho-polygons. *Mathematical Logic Quarterly 41*, 2 (1995), 261–267. 2

[Sha08] SHAMIR A.: A survey on mesh segmentation techniques. *Computer Graphics Forum 27*, 6 (2008), 1539–1556. 1, 2

[SR95] SHAPIRA M., RAPPOPORT A.: Shape blending using the star-skeleton representation. *Computer Graphics and Applications, IEEE 15*, 2 (mar 1995), 44 –50. 9

[STK02] SHLAFMAN S., TAL A., KATZ S.: Metamorphosis of polyhedral surfaces using decomposition. In *Computer Graphics Forum* (2002), pp. 219–228. 2

[TOS] TOSCA(TOOLS FOR NON-RIGID SHAPE COMPARISON AND ANALYSIS) PROJECT 3D DATASETS: *http://tosca.cs.technion.ac.il/book/resources_data.html*. 4

[XHH*10] XIA J., HE Y., HAN S., FU C.-W., LUO F., GU X.: Parameterization of star-shaped volumes using green's functions. In *Geometric Modeling and Processing* (2010), pp. 219–235. 1

[ZST*10] ZHENG Q., SHARF A., TAGLIASACCHI A., CHEN B., ZHANG H., SHEFFER A., COHEN-OR D.: Consensus skeleton for non-rigid space-time registration. *Computer Graphics Forum 29*, 2 (2010), 635–644. 9