Eurographics Symposium on Point-Based Graphics (2007)
M. Botsch, R. Pajarola (Editors)

Approximate Star-Shaped Decomposition of Point Set Data

Jyh-Ming Lien [†]

George Mason University, Fairfax, Virginia, USA

Abstract

Simplification or decomposition is a common strategy to handle large geometric models, which otherwise require excessive computation to process. Star-shaped decomposition partitions a model into a set of star-shaped components. A model is star shaped if and only if there exists at least one point which can see all the points of the model. Due to this interesting property, decomposing a model into star-shaped components can be used for computing camera locations to guard a given environment (the art-gallery problem), skeleton extraction, point data compression, as well as motion planning. In this paper, we propose a simple method to partition (or cluster) point set data (PSD) into "approximately star-shaped" components. Our method can be applied to both 2D and 3D PSD and can be naturally extended to higher dimensional spaces. Our method does not require or compute any connectivity information of the input points. The proposed method only requires the position and the outward normals of points. Our experimental results show that the size of the final decomposition is close to optimal.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computing Methodologies]: Computer Graphics[Computational Geometry and Object Modeling]

1. Introduction

Larger and more complex models can be captured cheaply nowadays using laser range scanners or stereo cameras. Large geometric models require a lot of computation resources to process. Even with current CPU and GPU advances, simplification is usually required in order to handle such models efficiently. Decomposition is a kind of simplification that partitions a model into simpler-shaped and smaller-sized components.

Star-shaped decomposition partitions a model into a set of star-shaped components. A model is star shaped if and only if there exists at least one point which can *see* all the points of the model. Due to this interesting property, decomposing a model into star-shaped components has many applications. For example, star-shaped decomposition can be used to extract shape descriptors, e.g., a skeleton. Star-shaped decomposition has been used to compute roadmaps and find paths in robotic motion planning [VM05]. Star-shaped decomposition is also closely related to the art-gallery problem (see [Chv75]): Finding the fewest cameras to guard a given environment.

In this paper, we propose a simple method to partition (or cluster) a point set into "approximately star-shaped" components. This work provides an alternative approach to the existing star-shaped decomposition methods that are mostly focused on decomposing 2D shapes with continuous boundaries. Our work is inspired by more and more techniques proposed to work directly on point set data (PSD) instead of on reconstructed meshes, e.g., rendering [RL00, ABCO*03], compression, feature extraction [PKG03], and surface analysis [PG01], mesh offsetting [CWRR06], to name just a few. One of the reasons for the popularity of point-based methods is that the connectivity of the points is not always easy to compute. An important benefit of working directly on points is that our method can be applied to both 2D and 3D point sets and can be naturally extended to higher dimensional spaces. Our method does not require or compute any connectivity information of the input points. The proposed method only requires the positions and the outward normals of points, which can usually be computed efficiently for data obtained from many sources, such as laser scanner or stereo cameras. Another important feature of the proposed method is that, as shown in our experimental results, our decomposition is close to optimal. In all of our experiments, the size of the final decomposition is only 1.1 to less than five times larger than the optimal size.

† jmlien@gmu.edu

We begin our discussion by reviewing some of the related work in Section 2. In Section 3, we define the terms and properties that we will use throughout the paper. The main algorithm of our decomposition method is presented in Sections 4 to 6. Finally, we show experimental results of the proposed algorithm in Section 7.

2. Related Work

Star-Shaped Decomposition. There is little known about 3D star-shaped decomposition. On the contrary, decomposing simple polygons into 2D star-shaped subpolygons is well studied; see a survey in [Kei00]. Similar to most of the decomposition problems, decomposing a polygon with holes into minimum number of star-shaped components is NP-complete [Kei83]. For polygons without holes, the partitioning can be done in $O(n\log n)$ time [AT81] or $O(n)$ [Gho83] time and result in $\frac{n}{3}$ components. Finally, Keil [Kei85] achieved the optimal solution for star-shaped partition with the minimum number criterion in $O(n^5 r^2 \log n)$ time, where r is the number of the reflex vertices.

Star-shaped decomposition is related to guarding an art gallery [Chv75]. A polygon is said to be guarded if it is covered by the visible regions of the guards. The visible region of a guard is a star-shaped component. The problem of finding minimum guards is known to be NP-complete for polygons with or without holes [OS83, LL86]. Approximate approaches also have high complexities, e.g., $O(n^5 \log n)$ time [Gho87] and $O(n^4 \log n)$ time [AGS88]. In 3D, approximate approaches have been proposed to guard terrain (see [BMKM05]).

Decomposition of Points. Several methods have been proposed to decompose point set data into meaningful components, e.g., by Dey et al. [DGG03]. Recently, Yamazaki et al. [YNBH06] proposed a decomposition method based on the estimation the centrality of each point using approximated geodesic distance. Despite their promising results, one major drawback of this approach is that the computation for the centrality is time consuming and requires large memory consumption. Further simplification of the model and approximation of the centrality are required to provide reasonable computation time.

Approximated Decomposition. The main idea of approximated decomposition is to take advantage of ignoring detailed features of the model to efficiently produce smaller decompositions than the exact approach. Approximate convex decomposition (ACD) of polygons [LA06] and of polyhedra [LA07] are methods that decompose a model into nearly convex components. Lien and Amato have shown that ACD can be generated more efficiently and has more applications than exact convex decomposition methods. As we will show, the approximate star-shaped decomposition has similar benefits.

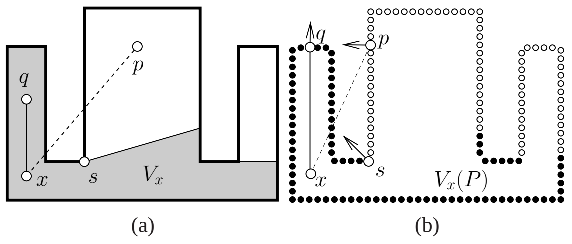


Figure 1: *(a) A polygon. (b) A point set representing the same shape. See Section 3 for detail.*

3. Preliminaries: Visibility

Before going into details about our decomposition method of points, let us first review some simple geometric properties about shape. A shape S is usually represented by its boundary ∂S and the boundary of a shape is usually represented by a set of connected points, i.e., edges and facets. Examples include 2D polygons and 3D polyhedra. Several geometric properties can be easily computed using boundary representations. For example, given two points p and q, we can check if p can directly *see* q (with respect to S) by checking if the line segment $\overline{pq}$ connecting p and q interests the boundary ∂S. That is we can define the visibility of two points as follows.

Definition 3.1. ***Visibility***. *Let p and q be two points and let S be a shape. Points p and q are visible from each other if and only if $\overline{pq}^{\circ} \cap \partial S \equiv \emptyset$, where $\overline{pq}^{\circ}$ is the open set of $\overline{pq}$.*

Now, given a point p inside a shape S, we call a set of points that are visible from p the *visible region* V_p of p (see Figure 1(a)).

Definition 3.2. ***Visible region***. *Let $p \in S$ be a point of S. The visible region, denoted as V_p, of p is a set of points in S that are visible from p, i.e., $V_p = \{q \in S \mid \overline{pq}^{\circ} \cap \partial S \equiv \emptyset\}$.*

It is easy to see that the visible region V_p of p is a star shape, in which the point p is visible from all the points of V_p. Therefore, we can now simply define a star shape and star-shaped decomposition using the concept of visible region.

Definition 3.3. ***Star shape and star-shaped decomposition*** *Shape S is a star shape if and only if there exists a point $p \in S$ so that $V_p \equiv S$. A star-shaped decomposition of a shape S is a set of star shapes $\{S_i\}$ whose union is S.*

Note that, so far, our definitions of visibility and star shape depend on a *continuous* boundary representation of the shape, however, this continuous boundary will not be available in a point-based representation.

Point-based representation. Now let us return our attention back to point sets. A point set data is also a type of boundary representation except that these points are *not* connected into lines or meshes. Without connectivity, the boundary ∂S of a shape S becomes more ambiguous, thus the definitions and properties mentioned previously seem to be not valid anymore. For example, how can we check if two points are visible from each other without the explicit representation of the boundary? Fortunately, as we will see, these properties can be *approximated* closely without computing the connectivity of the points.

We let P be a point set. We assume that the point set P is a sample of the boundary ∂S of a shape S and each point p of P is associated with an outward normal direction $\vec{n}_p$.

First, let's consider the visibility. Let x be an arbitrary point and p be a point of P. We observe that if the "viewing line" from x to p and the outward normal of p point in the opposite direction, p must be invisible from x. In this case, p is called a *back point* of x.

Definition 3.4. ***Back point****. Given a point x and a point $p \in P$. The point p is invisible from x if the normal of p is pointing in the opposite direction of $\overrightarrow{xp}$, i.e., $\vec{n}_p \cdot \overrightarrow{xp} < 0$, except the discontinuous points* [†].

Note that Definition 3.4 provides a necessary condition to identify invisible points of a guard x but not a sufficient condition. In order to find (or approximate) all visible points, we have to identify the points that are "occluded" by the back points as well. More precisely, we define the ε-view of a guard x as follows.

Definition 3.5. ***ε-view****. Given a guard x, an ε-view from x to a point p is represented as a cone whose apex is x, apex angle is ε and base center is p. Any back point of x in x's ε-view can occlude the view.*

Then we can immediately define the occluded points of a guard.

Definition 3.6. ***Occluded points****. A point p is occluded from a guard x if and only if there exists a back point q in x's ε-view and q is closer to x than p is to x.*

An example of the the ε-view is shown in Figure 2. This definition of ε-view helps to identity occluded points and therefore the visible points. Using the concept of ε-view, we define ε-visibility.

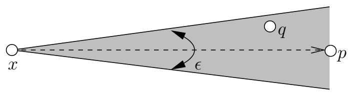


Figure 2: *An ε-view of a guard x towards a point p. If the point q is a back point of x, the view is occluded and p is not visible by x.*

Definition 3.7. ***ε-visibility****. A point p is ε-visible from a point x if p is neither a back point nor an occluded point of x. Then, a set of ε-visible points is called the ε-visible region of x, denoted as ε-V_x.*

Finally, we say that a point set P is an ε-approximate star shape (or simply ε-A*) if there exists a point x whose ε-visible region is P. Then, as before, an ε-A* decomposition of P is simply a set of ε-A*s whose union is P.

Note that the quality of the approximation of the true visibility depends on the value of ε. The quality degrades when ε is too large or too small. We will discuss how to compute ε from the point set in Section 6. In the following section, we will assume that ε is given by the user and start to sketch our method for computing A* decomposition.

4. Approximate Star-Shaped (A*) Decomposition

The framework of the A* decomposition is rather straightforward. We first select a point x at random from the point set P that is not visible from any existing guards. Then we add x as a guard and x's ε-visible region as an ε-A*to the decomposition. The process is iterated until all points in P are ε-visible by at least a guard. We pick this strategy because of its simplicity and because it has been shown to produce impressive results for polygons [AMP05]. More importantly, as we will see in our experimental results in Section 7, this strategy generates only 1.1 to less than 5 times more guards than any optimal solutions do for some common models in computer graphics (see Table 1).

The main challenge of our A* decomposition framework is to approximate the visible region of a guard. Because of its crucial role in this framework, we will spend the entire next section addressing this challenge.

5. ε-Visible Region of A Guard

In this section, we will discuss methods to compute the ε-visible region of a guard. We first propose a basic method in Section 5.1. Then we will improve of the efficiency and quality of this basic method in Section 5.2 and in Section 5.3, respectively.

5.1. Basic Approach

This basic approach is composed of two steps: Radial partitioning and ε-visible point identification.

Radial partitioning. Given a point set P and a guard x, we use x to partition P. To do so, we convert the points of P to a spherical coordinate system centered at x. Then, we partition the spherical coordinate system into $2(\frac{\pi}{\varepsilon})^2$ equally-sized buckets. Each bucket represents an ε-view of x. Finally, each point of P is assigned to a bucket according to its coordinate. In our implementation, we use an enclosing box instead of a sphere, and we project P to the boundary of the box. An example of such a partitioning is shown in Figure 3(a). A benefit of using a box is its simplicity in extending to high dimensional space.

Find visible points. We identify ε-visible points from each bucket. That is, for each bucket, we find the closest back point p of x (see the shaded bucket in Figure 3(b)). Then all points in the bucket that are closer than p to x are

[†] Points on the boundary of a visible region can be further classified into connected components. We call the boundary points of each component "discontinuous points." An example is the point s in Figure 1(b). The discontinuous points must be reflex vertices (whose internal angle is larger than 180°). The number of discontinuous points is usually relatively small comparing to the the number of the invisible points.

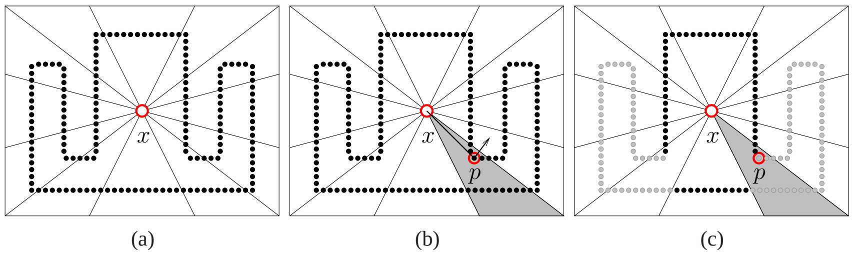


Figure 3: *(a) A radial partitioning of the point set P from a guard x. Points in P are projected to the boundary of a box defined around x. (b) For each bucket, the closest invisible point p is found and all points that are closer than p are considered visible. (c) Darker points are considered as visible points from x, which collectively form an* A* *of x.*

visible by x. Figure 3(c) shows all the ε-visible points from x, which collectively form the A* of x. Algorithm 5.1 outlines this process.

Algorithm 5.1: A$^\star(P, x, \varepsilon)$

```
comment: Point set $P$ and query point $p$
Assign $P$ into $2(\frac{\pi}{\varepsilon})^2$ buckets $\{B_i\}$ of $x$'s radial partitioning
for each bucket $B_i$
  do  Compute the closest back point $p \in B_i$ to $x$
      if $p$ exists
        then $d_p \leftarrow |x - p|$
        else $d_p \leftarrow +\infty$
      for each point $r \in B_i$
        do  if $|x - r| < d_p$
              then $V_x \leftarrow V_x \cup r$
```

A naïve implementation of Algorithm 5.1 is of $O(n)$ time complexity for a point set with n points. The computation efficiency can be further improved if a spatial data structure is built on P. Details of this idea will be elaborated in Section 5.2. Moreover, since the guard x is selected at random, x can have poor visibility. In this case, more guards will be needed to guard the entire point set. Therefore, in order to reduce the number of guards needed to cover the space, we find a "better x" from x. Details of this method will be discussed in Section 5.3.

5.2. Improve A* Efficiency: Box-tree preprocessing

We use a **box tree** to pre-process the input point set and to improve the performance of the A* computation discussed above. A box tree is similar to an octree except that, in the box tree, each node is always a smallest (axis-aligned) bounding box of the enclosed points. Intuitively, our plan is to place points near a guard x in smaller boxes and place far away points in larger boxes. With this data structure, we hope we can find the closest back point in the nearby small boxes without checking the far away large boxes.

Box tree construction. The construction of the box tree starts with the minimum bounding box of the entire point set. If there are more than k points in a box, these points are partitioned evenly at their center into eight point sets. (In our experiment we arbitrarily pick $k = 16$.) Then a minimum bounding box is constructed for each point set as a child of the original box. This process iterates until no boxes can be split.

Radial partitioning. Similar to the radial partitioning of the entire point data, we perform a radial partitioning on the box tree constructed above. The idea is to traverse the box tree top-down and find a set of boxes that can fit into the buckets. Starting with the bounding box (the root), we check if all of its vertices belong to the same bucket. If so, we assign the box to the bucket. Otherwise, we "open" the box and perform the same test for each of the eight child boxes and repeat until no unfit boxes left.

Find visible points. Similar to the basic method, we find the closest back point in each bucket. To do so, we first sort the boxes from near to far and then start to examine points in each box until we find a back point.

Since checking if a box fits into a bucket takes only constant time and the number of boxes is much smaller than the number of points, partitioning boxes is more efficient than partitioning all points. Using a box tree, the complexity of computing an A* becomes output sensitive, i.e., $O(k+m)$, where k is the number of visible vertices and m is the overhead of building the box tree and sorting the boxes in each bucket.

5.3. Improve A* Quality: A* Expansion

Our goal here is to find a better guard than the randomly selected guard. Here, "better" means larger visible region. Our strategy is to compute the *kernel* of the visible points of a guard x and find a guard from the kernel. We define a kernel of a point set as the following.

Definition 5.1. *A kernel of a set points P is another set of points K. Every point of K can see all the points of P.*

The kernel K_p of a set of points V_p can be computed as the intersection of half-spaces defined by the points in V_p (see Figure 4). The key to compute the kernel efficiently is that we can convert these half-spaces to the points $\overline{V_p}$ in the

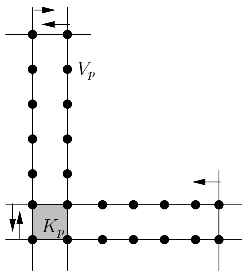


Figure 4: *K_p is the kernel of the points V_p. K_p is simply the intersection of the half spaces (shown as the lines and arrows in the figure) defined by the points (shown as dark points in the figure) and point normals.*

dual space, where $\overline{V_p}$ is defined as the following:

$$\overline{V_p} = \{\frac{\vec{n}_q}{\overrightarrow{qp} \cdot \vec{n}_q} \mid q \in V_p\}.$$

Then the boundary points in the kernel K_p are simply dual to the facets of the convex hull of $\overline{V_p}$, which can be computed efficiently in $O(n\log n)$, where n is the number of points in V_p.

Lemma 5.2. *Let $H_i = \{x \mid (x - p_i)\cdot \vec{n}_{p_i} < 0\}$ be a half space defined by a point $p_i \in V_p$. The kernel K_p of V_p is $\bigcap_i H_i$.*

Proof. We know that if a point $p_i \in V_p$ is visible from a point x, the view direction $\overrightarrow{xp_i}$ and the normal of p_i point in the same direction. For x to be a member of the kernel K_p, this criterion must hold for all points of V_p, i.e., $\overrightarrow{xp_i}\cdot\vec{n}_{p_i} > 0 = (x-p_i)\cdot\vec{n}_{p_i} < 0, \forall p_i$, i.e., $\bigcap_i H_i$. □

Once the kernel is known, we compute a visible region for each vertex in the kernel. Since the visible regions of the vertices of the kernel are larger than or equal to $V_p(P)$ (see Lemma 5.3), our visible region must expand monotonically. This process is repeated until no expansion can be gained. The subroutine A*-EXPAND is defined in Algorithm 5.2.

Algorithm 5.2: A*-EXPAND(x, V_x)

```
Compute the kernel $K_x$ from $V_x$
Find $k \in K_x$ so that $k$ has the largest visible region $V_k$
if $|V_k| > |V_x|$
  then return (A$^\star$-EXPAND$(k, V_k)$)
  else return $(x, V_x)$
```

Lemma 5.3. *Given a point p and its visible points V_p, Algorithm 5.2 must return a guard p' whose visible region is no smaller than V_p.*

Proof. Let K_p be the kernel of V_p and let $p' \in K_p$. Because p' can see all the points in V_p, $V_p \subset V_{p'}$. Therefore $|V_p| \le |V_{p'}|$. □

It is clear that the time complexity of computing A*-EXPAND is higher than computing A*. One pass of A*-EXPAND without the recursive call takes $O(n^2 + n\log n)$ time. In the worst case, only one more point becomes visible by applying one pass of A*-EXPAND. When this happens, the recursion depth is $O(n)$ and the total time complexity for A*-EXPAND is $O(n^3)$, which is not practical for most problems that we are interested in in this work. To reduce the time complexity, we modify Algorithm 5.2 so that only $\log|K_p|$ points randomly selected from the kernel K_p are considered and the recursion stops when the improvement is less than $|P|/c$ points, where c is a user-defined constant. This heuristic is outlined in Algorithm 5.3.

Algorithm 5.3: A*-EXPAND2(x, V_x)

```
Compute the kernel $K_x$ from $V_x$
Let $K'_x$ contain $\log|K_x|$ random vertices from $K_x$
Find $k \in K'_x$ so that $k$ has the largest visible region $V_k$
if $|V_k| > |V_x| + \frac{|P|}{c}$
  then return (A$^\star$-EXPAND2$(k, V_k)$)
  else return $(k, V_k)$
```

The time complexity of Algorithm 5.3 now becomes $O(n\log n)$. Experimentally we observe that A*-EXPAND and A*-EXPAND2 produce similar results while A*-EXPAND2 is much more efficient. For the rest of this paper, A*-EXPAND2 is used (if not said otherwise) to compute the A* of a given point.

6. Putting It All Together

Algorithm 6.1 shows a fleshed-out version of the A* decomposition. Note that in the last step of Algorithm 6.1 we simple assign each point to its closest visible guard to produce the final decomposition. Figure 5 shows an example of the A* decompositions of a 3-d point cloud. Algorithm 6.1 has $O(kn\log n)$ time complexity, where n and k are the number of points in P and guards, respectively.

Algorithm 6.1: A*-DECOMP(P)

```
build a box tree from $P$
repeat
  randomly select a point $x \in P$ invisible by any guards
  build $\varepsilon$-$V_x$ of $x$ using the box tree
  $(g, \varepsilon\text{-}V_g)$ =A$^\star$-EXPAND2$(x, \varepsilon\text{-}V_x)$
  add $(g, \varepsilon\text{-}V_g)$ to the decomposition
until every point in $P$ is visible
Assigneverypointtoitsclosestvisibleguard
```

Now, the remaining task is to find the value of ε from a given point set. As we mentioned before, the value of ε has great influence on the quality of the final decomposition. A small ε makes invisible points visible while a large ε eliminates many visible points. Our goal is to find the smallest ε that will not classify invisible points as visible.

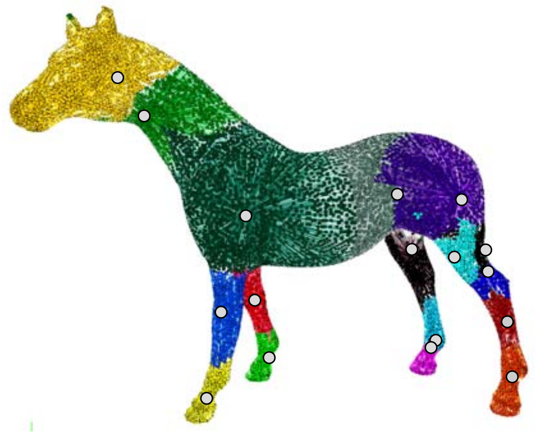


Figure 5: A* *decompositions of the point data of a horse model (with* A*-EXPAND2*). Each of the 17 large dots in the figure represents a guard. (This figure is much informative in color. Please refer to the PDF file if you have a black and white printout.)*

6.1. The value of ε

The value of ε depends on how dense the point set P is sampled from the boundary of a shape S and the *external* medial axis of S. Lemma 6.1 uses the sampling density to compute the value of ε.

Lemma 6.1. *When* $\varepsilon = \max_i\{2\cdot\arctan(\frac{\delta}{4\gamma_i})\}$, *where* δ *is the sampling density of* P *and* γ_i *is the distance from a point* $p_i \in P$ *to its closest point on the medial axis, Algorithm 6.1 will never classify invisible points as visible points.*

Proof. We show this statement is true by assuming that ε is known and show that if no invisible point is missed by Algorithm 6.1, then the sampling density is $4\gamma_i\cdot\tan(\frac{\varepsilon}{2})$. Figure 6 shows an example of the worst scenario, in which the guard x is placed on the boundary of the shape. In addition, x's ε-view is aligned with its outward normal and the first segment of the shape blocking the view is perpendicular to the view. This make the intersection of the view and the segment shortest.

To make Algorithm 6.1 recognize this boundary, there must be a point sampled from the intersection, e.g., the point p in Figure 6. Otherwise, x will see through this boundary and classy some invisible points as visible, e.g., the point q. Therefore the length of the intersection of the ε-view and the invisible boundary defines the sampling density δ. It's easy to see that the length of the intersection is $4\gamma_x\cdot\tan(\frac{\varepsilon}{2})$, where γ_x is the distance between the external medial axis and x. Thus, we conclude that if $\varepsilon = 2\cdot\arctan(\frac{\delta}{4\gamma_x})$ all back points will be found and no invisible points are visible by x. □

Both sampling density δ and distance to the medial axis γ_i can be estimated from the point set. To estimate, δ, we compute the maximum of the longest distances between each

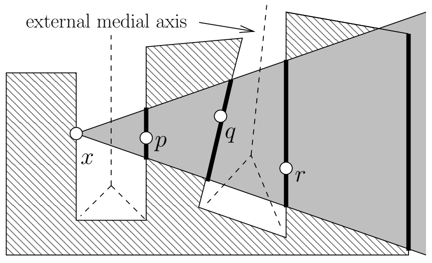


Figure 6: *There must be a sample, such as the point* p, *in the first intersection of* x*'s* ε*-view and the polygon to ensure that* q *will not become a visible point of* x.

point and its k nearest neighbors. (We simply set $k = 4$.) We estimate γ_i as the half of the distance between the point p_i and its closest back point.

7. Experimental Results

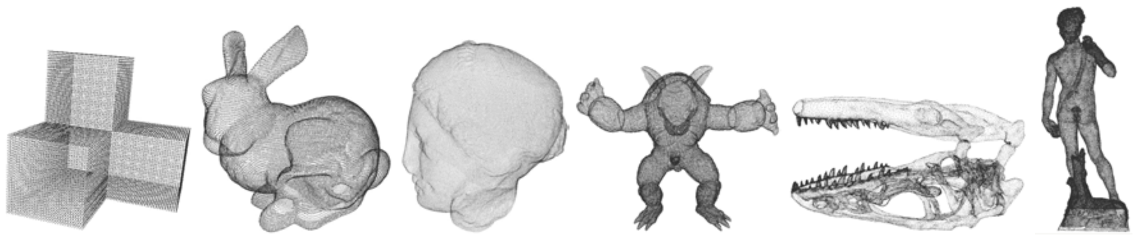
We implemented the proposed method in C++. In this section, we show experimental results from our implementation using seven common models, which are converted to point set data and shown in Table 1. We show the the number of guards and computational efficiency of the proposed methods. We also compare to the estimated lower bound of guards for each point set.

The size of A* decomposition is close to the lower bound. We compute the lower bound as the maximum number of guards such that no points can see more than one guard. Under this definition, we know that no visible regions of any guards can overlap and therefore this lower bound must be smaller than the minimum number of guards covering the entire point set.

The estimated lower bound for each point set is shown in Table 1. Table 1 also shows the averaged number of guards generated by A* decomposition over 10 runs. For the cubes, horse, bunny, venus and armadillo models, A* decomposition generates just 1.1 to 2.7 times more guards than the lower bound does. Moreover, when the shape of the model is relatively "fat" (e.g., the cubes, the bunny and the venus models), A* decomposition generates only 1.1 to 1.5 times more comparing to the lower bound. On the other hand, A* decomposition generates 2.7 times and 4.9 times more guards than the lower bound of the head bone and the david models. The main reason for this is because the lower bound estimation is *not* accurate enough. This happens when the model has "teeth"-like or pocket-like regions, e.g., the teeth of the bone model and the pockets around the hair area of the david model. A guard placed at the based of the "teeth" will prohibit any guards being placed inside the teeth. Figure 7 illustrates this scenario.

A* expansions reduce decomposition size. Recall that, in Section 5.3, we proposed two A* expansions (denoted as full and limited expansions). A* expansion is designed to improve the visibility of a randomly selected guard and to

Table 1: *Models used in the experiments.* A* *decompositions of the models are generated with limited expansion.*



		cubes	horse	bunny	venus	armadillo	head bone	david
	point set	(above)	(Fig. 5)	(above)	(above)	(above)	(above)	(above)
	point size	18434	19849	34834	134345	172974	172974	254072
number of guards	A*-DECOMP (avg.)	1.1	18	6.5	2.3	40.4	276.2	235
	lower bound	1	12	4	2	22	109	48

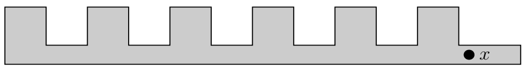


Figure 7: *In the lower bound estimation, the guard x will prohibit us from adding more guards, while the minimum number of guards to cover the environment is seven.*

reduce the size of the final decomposition. Figure 8 shows the computation time and the number of guards. For the limited expansion, we stop expanding when the expansion is not larger than 0.1% of the input point size.

As you can see from Figure 8, the number of guards generated without any expansion is significantly (2 to 5 times) larger than that with expansions. Moreover, the difference between the guard sizes of the full and the limited expansions is negligible while the limited expansion is always (1.3 to 3 times) faster than the full expansion.

Box tree helps improving efficiency. The box tree described in Section 5.2 significantly improves the efficiency. Figure 9 shows that the improvement is larger than 100%.

Partial coverage. When partial visibility coverage is tolerable, A* decomposition can be computed much more efficiently. In fact, according to Figure 10, if only 90% coverage is needed the number of guards and the computation time can be halved. Both the size of guards and the computation time grow exponentially as the coverage percentage increases.

8. Conclusions

In this paper, we proposed a method to decompose a point set into a set of approximately star-shaped (A*) components. We proposed a method to compute ε-visibility, where ε can be estimated from the input point set. Several strategies are investigated to improve the quality and the efficiency of the A* decomposition, namely the box tree and A* expansion. Our experiments demonstrated these improvements using seven common models.

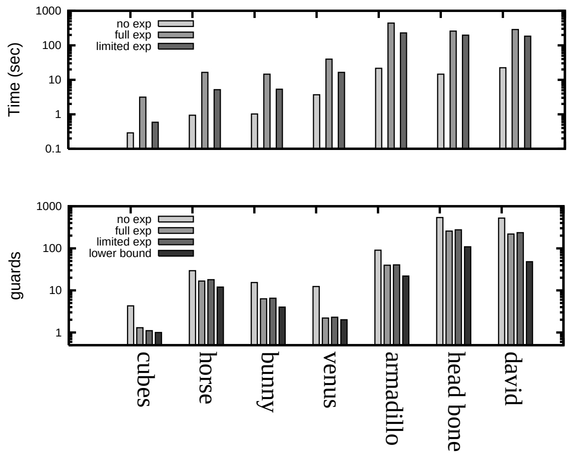


Figure 8: *Computation time and guard size with three different* A* *expansions. Each record shows an average of 10 experiments. Notice that the Y-axis is in logarithmic scale.*

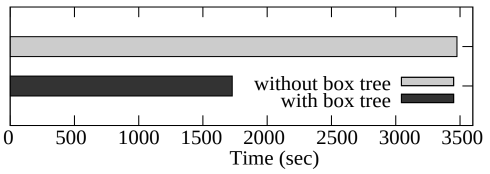
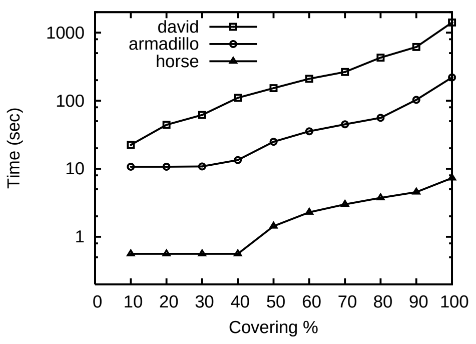


Figure 9: *Decomposition time of all seven models with and without box trees.*

The major limitation of the proposed method is its efficiency. The computation can take up to several minutes (4.6 and 3.9 minutes are needed for the bone and the david models, respectively). Fortunately, due to the popularity of the recent multi-core processors, we can address this issue by



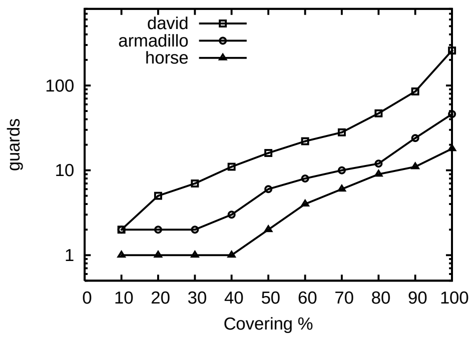


Figure 10: *Computation time and number of guards needed for different percentages of coverage. Three models are shown in this experiments. Similar behaviors are observed for all three models. Notice that the Y-axis is in logarithmic scale.*

parallelizing the proposed method. This can be done by asking processors to work on different buckets independently.

References

[ABCO*03] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics 9*, 1 (2003), 3–15.

[AGS88] AGGARWAL A., GHOSH S. K., SHYAMASUNDAR R. K.: Computational complexity of restricted polygon decompositions. In *Computational Morphology*, Toussaint G. T., (Ed.). North-Holland, Amsterdam, Netherlands, 1988, pp. 1–11.

[AMP05] AMIT Y., MITCHELL J., PACKER E.: On guarding and partitioning polygons. In *15th Annual Fall Workshop on Computational Geometry and Visualization* (2005).

[AT81] AVIS D., TOUSSAINT G. T.: An efficient algorithm for decomposing a polygon into star-shaped polygons. *Pattern Recogn. 13* (1981), 395–398.

[BMKM05] BEN-MOSHE B., KATZ M. J., MITCHELL J. S. B.: A constant-factor approximation algorithm for optimal terrain guarding. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA, 2005), Society for Industrial and Applied Mathematics, pp. 515–524.

[Chv75] CHVÁTAL V.: A combinatorial theorem in plane geometry. *J. Combin. Theory Ser. B 18* (1975), 39–41.

[CWRR06] CHEN Y., WANG H., ROSEN D. W., ROSSIGNAC J.: A point-based offsetting method of polygonal meshes, 2006. ASME Journal of Computing and Information Science in Engineering, in review.

[DGG03] DEY T. K., GIESEN J., GOSWAMI S.: Shape segmentation and matching with flow discretization. In *Proc. Workshop on Algorithms and Data Structures* (2003), pp. 25–36.

[Gho83] GHOSH S. K.: A linear time algorithm for decomposing a monotone polygon into star-shaped polygons. In *Proc. 3rd Conf. Found. Softw. Tech. Theoret. Comput. Sci.* (1983), pp. 505–519.

[Gho87] GHOSH S. K.: Approximation algorithms for art gallery problems. In *Proc. Canadian Inform. Process. Soc. Congress* (1987), pp. 429–434.

[Kei83] KEIL J. M.: *Decomposing Polygons into Simpler Components.* PhD thesis, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, 1983.

[Kei85] KEIL J. M.: Decomposing a polygon into simpler components. *SIAM J. Comput. 14* (1985), 799–817.

[Kei00] KEIL J. M.: Polygon decomposition. In *Handbook of Computational Geometry*, Sack J.-R., Urrutia J., (Eds.). Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000, pp. 491–518.

[LA06] LIEN J.-M., AMATO N. M.: Approximate convex decomposition of polygons. *Comput. Geom. Theory Appl. 35*, 1 (2006), 100–123.

[LA07] LIEN J.-M., AMATO N. M.: Approximate convex decomposition of polyhedra. In *SPM '07: Proceedings of the 2007 ACM symposium on Solid and physical modeling* (New York, NY, USA, 2007), ACM Press, pp. 121–131.

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

[OS83] O'ROURKE J., SUPOWIT K. J.: Some NP-hard polygon decomposition problems. *IEEE Trans. Inform. Theory IT-30* (1983), 181–190.

[PG01] PAULY M., GROSS M.: Spectral processing of point-sampled geometry. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2001), ACM Press, pp. 379–386.

[PKG03] PAULY M., KEISER R., GROSS M.: Multi-scale feature extraction on point-sampled surfaces. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing* (2003), pp. 281–289.

[RL00] RUSINKIEWICZ S., LEVOY M.: Qsplat: a multiresolution point rendering system for large meshes. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2000), ACM Press/Addison-Wesley Publishing Co., pp. 343–352.

[VM05] VARADHAN G., MANOCHA D.: Star-shaped roadmaps - a deterministic sampling approach for complete motion planning. In *Robotics Science & Systems* (2005).

[YNBH06] YAMAZAKI I., NATARAJAN V., BAI Z., HAMANN B.: Segmenting point sets. In *IEEE Intl. Conf. Shape Modeling and Applications (SMI)* (2006), pp. 4–13.