Building Binary Orientation Trees from Unorganized Point Clouds through Visibility Checks

陳奕麟
國立清華大學
yilin@cs.nthu.edu.tw

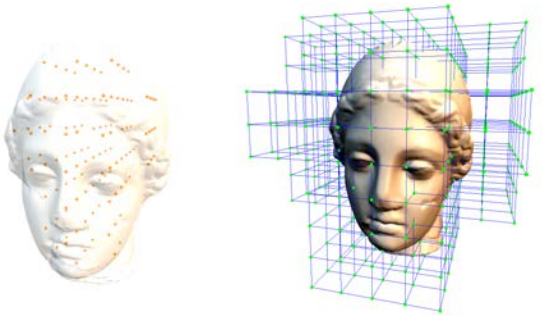
陳炳宇
國立台灣大學
robin@ntu.edu.tw

賴尚宏
國立清華大學
lai@cs.nthu.edu.tw

西田友是
東京大學
nis@is.s.u-tokyo.ac.jp

ABSTRACT

Given a complete unoriented point set, we propose a binary orientation tree (BOT) for volume and surface representation, which roughly splits the space into the interior and exterior regions with respect to the input point set. The BOTs are constructed by performing a traditional octree subdivision technique while the corners of each cell are associated with a tag indicating the **in**/**out** relationship with respect to the input point set. Starting from the root cell, a growing stage is performed to efficiently assign tags to the connected empty sub-cells. The unresolved tags of the remaining cell corners are determined by examining their visibility via the hidden point removal operator. We show that the outliers accompanying the input point set can be effectively detected during the construction of the BOTs. After removing the outliers and resolving the **in**/**out** tags, the BOTs are ready to support any volume or surface representation techniques. To represent the surfaces, we also present a modified MPU implicits algorithm enabled to reconstruct surfaces from the input unoriented point clouds by taking advantage of the BOTs.



(a) (b)

Figure 1: The BOT splits the space into inside/outside and assists the surface reconstruction process from the Igea data set. The BOT is rendered by the leaf nodes of level 3 with the corners tagged as in (shown in (a) by the orange points) and out (shown in (b) by the green points), respectively.

1. INTRODUCTION

Hierarchical space partitioning structures, such as octrees [9], Binary Space Partitioning (BSP) trees [4] and k-d trees [2], are extensively exploited in various research fields. The simplicity of constructing such structures makes them very popular in many computer graphics applications in order to achieve better efficiency or to be scaled toward large data sets. Most of the existing hierarchical structures put emphasis on partitioning the space to produce a collection of subsets of the data satisfying a given error criterion. However, few of them attempted to combine additional semantic information to assist the processing of the input data sets.

In this paper, we present the *binary orientation trees* (BOTs) for representing unoriented and unstructured point clouds as well as the semantic metadata for representing their volumes and surfaces, A BOT roughly splits the space into two parts, i.e. *inside* or *outside*, by exploiting the observation that any point not belonging to a model can only be either *visible* or *invisible* from various viewpoints. Given a *complete* point set, a BOT is built by performing the traditional octree-based space partitioning technique while the corners of each cell are associated with a tag indicating the **in**/**out** relationship with respect to the input point set.

Conceptually, the BOTs are similar to the *signed distance fields* derived from an explicit or implicit surface representation. Although it is simple to perform in/out checks if an implicit surface or polygonal mesh approximating the 3D model is given, to understand where the inside/outside part is with respect to a *discretely* sampled point cloud is not a trivial task, especially when the orientation information is not available. Besides the **in**/**out** information stored in the tree nodes as the traditional octree-based volume data representation, the BOT also contains the surface points sampled from the original model's surface or filtered from the original point cloud. Hence, the BOTs contain sufficient information for representing the volume and surface of the input point set and ready to support any volume or surface representation techniques.

The kernel of building the BOTs is a tagging algorithm that separate the exterior points of a raw point set from the interior ones. Starting from the root cell with all corners tagged as **out**, a *growing* stage is performed to efficiently assign tags to the connected empty sub-cells. Based on the observation that the cell corners are either visible or "occluded" by the input point cloud when viewed from outside, the untagged corners can be effectively classified into interior or exterior points by examining their visibility via the *hidden point removal* (HPR) operator [12].

It is worth noting that BOTs possess some advantages not shared by the traditional octree-based volume data representation. First of all, it can be directly computed from raw data points lacking of essential information like normal vectors. Since it keeps the original point set, it can be adaptively refined when necessary. Besides, the **in**/**out** information carried by a BOT can be exploited to reconstruct the model surface from the surface points. If the original input point cloud contains some outlier points, the outliers will be removed during the BOT construction.

The rest of this paper is organized as follows: we firstly review some related work in Section 2 and then present the algorithm of constructing the BOTs while removing the outliers in Section 3. In Section 4, we show the application to surface reconstruction from the input unoriented point sets through a modified MPU implicits algorithm [17] based on the BOTs. Sections 5 and 6 demonstrate the experimental results and conclude this paper, respectively.

2. RELATED WORK

Hierarchical space partitioning data structures are ubiquitous in computer graphics applications as the need of decomposing and structuring the input data sets. The basic approaches of building such structures are to recursively subdivide a root cell embedding a scene or an object until all leaf nodes satisfy a certain user-specified criterion. The simplest space partitioning strategies are perhaps the 1-to-4 scheme for quadtrees in 2D and 1-to-8 scheme for octrees in 3D [9, 19], or even uniform grids. Binary space partitioning (BSP) [4] is a generic process of recursively dividing a scene into two parts by a set of the best splitting hyperplanes. The k-d tree structure is a special case of the BSP trees and is constructed through orthogonal space separation [2]. Recently, Boubekeur *et al.* [3] developed the Volume-Surface Trees, which switch the standard octree subdivision to quadtrees as soon as the local area associated with the current node passes a height field test, in order to improve the imbalanced clustering of pure volume-based space decomposition.

Due to the nature of producing a hierarchy of data clusters, spatial partitioning techniques are widely applied for efficient data processing and manipulation. In geometric modeling and processing, surface reconstruction from point clouds has attracted considerable attentions [5, 7, 13]. The *partition-of-unity* approaches aim to reconstruct the surfaces formed by smoothly blended local implicit patches through adaptive octree [17] or BSP tree [20] approximation. Following the pioneering work of *Marching Cubes* algorithm [16], which extracts polygonal meshes from volume data, a number of iso-surface extraction algorithms have been developed for the purpose of feature preservation [11, 15] or topology consistency [6, 14].

In [10, 24], surface reconstruction was accomplished by identifying the exterior (interior) volumetric grid points, which is similar to the construction of the BOTs, in order to define an initial implicit surface. However, the regular grids used in these methods are associated with a scalar field derived from computing a distance transform by PDEs to guide the tagging process. Xie *et al.* [22] also proposed a regular volumetric structure called "*mono-oriented regions*" computed by active contour method, which carries similar semantic information as the proposed BOTs. They were utilized to determine the orientation of locally fitted implicit surfaces through a voting process. The drawbacks of such structures include that they are not adaptive to the feature sizes of input data sets and the high computational cost.

3. BINARY ORIENTATION TREES

3.1 Overview

A binary orientation tree (BOT) $\mathcal{T}$ is an octree-like hierarchical structure built from a point set $\mathcal{P}$ with all cell corners associated with a tag indicating the **in**/**out** relationship with respect to $\mathcal{P}$. Only the spatial positions of the data points are assumed to be available. To obtain the **in**/**out** information from a raw point set is a challenging problem without the underlying surface. The basic idea is to identify the interior and exterior points according to their visibility from various viewpoints. Hidden point removal (HPR) is a simple method that determines the visible points among a point set from a specific viewing direction through a spherical flipping transformation and convex hull computation [12]. By exploiting the HPR operator, the exterior points will be observed when viewed from outside, while the interior points will be occluded by $\mathcal{P}$. Before explaining the construction process, we first define some notations and examine some properties of BOTs in this section.

A cell of a BOT $\mathcal{T}$ is **mono-oriented** if the space enclosed by it is intrinsically the same, i.e. either inside or outside, otherwise it is called **bi-oriented**. It is obvious that an empty cell is certainly mono-oriented. A *maximal mono-oriented cell* is not contained by other mono-oriented cells and is *self-orientable* since the tags of the sub-cells can be directly determined by their parent cell. As a result, it is not necessary to perform further subdivision as soon as the current cell becomes mono-oriented during the construction of the BOT. It is worth noting that a cell with all corners associated with the same tag is not necessary to be mono-oriented. For example, the root cell of $\mathcal{T}$ is typically assigned a tag **out** for each corner. However, it is obvious that it consists of both the interior and exterior regions of $\mathcal{P}$.

3.2 BOT Construction

Given a complete point set $\mathcal{P}$, i.e. there exists no hole in $\mathcal{P}$, the construction of a BOT $\mathcal{T}$ consists of two main stages: *partitioning* and *tagging*.

- **Partitioning**: Initially, $\mathcal{P}$ is inserted into an axis-aligned bounding box as the root of $\mathcal{T}$. Then, $\mathcal{P}$ is decomposed into a collection of subsets $\mathcal{P}_i$ through the octree subdivision. We adopt the simple criteria of the

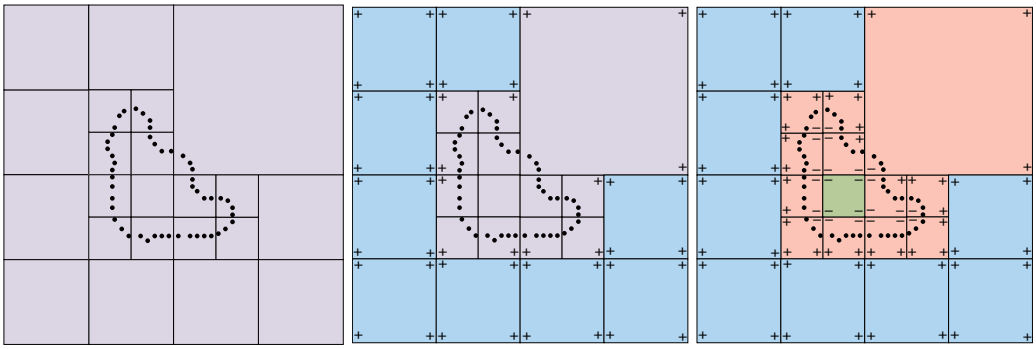


Figure 2: The construction process of a BOT. (a) Octree subdivision of a point set $\mathcal{P}$. (b) The growing process identifies a set of mono-oriented cells tagged as out (the blue cells with + tags). (c) The carving process resolves the remaining untagged corners and identifies the mono-oriented cells tagged as in (the green cell with − tags) and the bi-oriented cells (the red ones).

size of $\mathcal{P}_i$ as well as the *surface variation* [18] measuring if the points in $\mathcal{P}_i$ are isotropically distributed or well approximated to a local plane to guide the subdivision of $\mathcal{T}$. The recursion of the octree subdivision is stopped as soon as the current node becomes empty.

- **Tagging**: In this stage, each corner of a cell is classified as **in** or **out** through a tagging process. Exploiting the property of mono-oriented cells, once a corner of an empty cell is tagged, the tags of the rest corners can also be determined. As a result, a *growing* process is performed to efficiently tag the connected empty regions. For non-empty cells where the tags cannot be explicitly determined, a *carving* process is carried out to check the visibility of the untagged corners by using HPR.

Figure 2 illustrates an example of the BOT construction. Note that since HPR works in 2D, a BOT can be constructed in 2D as well by replacing the octree with quadtree.

3.2.1 Growing

To tag a sufficiently partitioned $\mathcal{T}$, we start by *growing* the mono-oriented cells surrounding $\mathcal{P}$ into regions. Apparently, the root of $\mathcal{T}$ should be assigned the **out** tag to each corner. Beginning with these corners explicitly tagged, the connected empty cells can also be explicitly tagged. Though this region growing problem can be resolved by some existing solutions like front propagation, we devise a recursive back-tracing tagging procedure, which can be easily realized in a tree structure like $\mathcal{T}$. Algorithm 1 depicts the pseudo-code for the growing process. For an intermediate node, the `tag_tree()` routine assigns the tags to its subtree(s) by back tracing from the leaves containing its tagged corners and propagating the tags if an empty cell is encountered. Note that the `back_tracing()` routine is launched if the currently visited node is a leaf or has been previously checked. The `tag_corners_if_empty()` routine sets all corners of an empty cell as long as one of them is tagged.

3.2.2 Carving

Algorithm 1 The pseudo-code of the growing process.

```
tag_tree(C)
begin
C' ← all leaves containing tagged corners of C;
if C is not leaf and not traced then
  for all cells C_leaf in C' do
    call back_tracing(C_leaf);
  end for
end if
if C is not leaf then
  for all subcells C_sub of C do
    call tag_tree(C_sub);
  end for
end if
end

back_tracing(C)
begin
call tag_corners_if_empty(C);
if C is not leaf then
  set C as traced;
  for all subcell C_sub of C do
    call tag_corners_if_empty(C_sub);
  end for
else if C is an intermediate cell then
  C_parent ← parent cell of C;
  call back_tracing(C_parent);
end if
end
```

After the growing process, most of the remaining untagged corners belong to the bi-oriented cells, where the point set $\mathcal{P}$ "pass" through. Among them, some are interior points occluded by $\mathcal{P}$. As a result, we determine the tags according to their visibility from various viewing directions by applying the following iterative process.

1. Collect the untagged corners in BOT to form $\mathcal{P}'$.
2. Perform HPR on $\mathcal{P} \cup \mathcal{P}'$ from a pre-selected viewpoint.
3. "Carve out" the visible points in $\mathcal{P}'$ and assign them

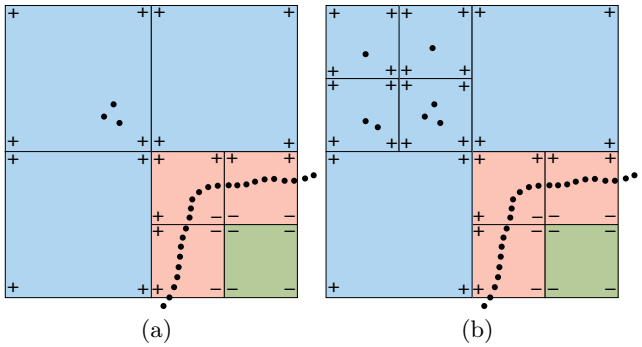


Figure 3: Outlier removal during the construction of the BOT. (a) An isolated outlier cell. (b) A cluster of the outlier cells.

an **out** tag.

4. Pick up the next viewpoint and repeat Step 2.

5. If no visible point belonging to $\mathcal{P}'$ can be detected, terminate the process. The remaining points in $\mathcal{P}'$ are assigned the **in** tag.

One reason that the tagged points in $\mathcal{P}'$ are excluded from the next iteration is that the points in $\mathcal{P}'$ sometimes also occlude each other according to the viewpoints though it is rare. Nevertheless, it is unnecessary to keep the observed and tagged points during each iteration.

The viewpoint selection with respect to various input point sets is very difficult, especially when some points lying within a concave region and can only be observed from some specific viewpoints. Due to the fact that $\mathcal{P}'$ somewhat occludes $\mathcal{P}$, if all points in $\mathcal{P}$ is observed during the carving process outlined above, it is very likely that all points in $\mathcal{P}'$ are also observed. We thus use a mask to record if a point in $\mathcal{P}$ has been observed or not. In the beginning of the carving process, a set of pre-defined viewing directions, such as the main axes of the bounding box of $\mathcal{T}$, can be used to carve out most exterior points in $\mathcal{P}'$. Since the unobserved points are usually clustered due to the occlusion from the previous viewing directions, we randomly choose an unobserved point in $\mathcal{P}$ and search for the nearest point in $\mathcal{P}'$ whose distance to each point $\mathbf{p} \in \mathcal{P}$ is larger than a threshold δ and already tagged as **out** as the new viewpoint to expose the unobserved regions. The purpose of δ is to prevent from selecting a viewpoint too close to $\mathcal{P}$, which may cause the line-of-sight to penetrate into $\mathcal{P}$ and disocclude the interior region. Obviously, the proposed tagging algorithm is limited to complete point clouds since the holes contained in $\mathcal{P}$ reveals its interior region when viewed from specific viewing directions.

3.3 Outlier Removal

In this subsection, we explain how outliers embedded in the input point cloud $\mathcal{P}$ can be effectively detected during the construction of the BOTs. Conceptually simple, the observation is that the outliers are usually sparse and disorderly distributed, and thus they can hardly occlude the cell corners surrounding them. During the tagging stage, the corners of the cells containing the outliers will most likely be observed. To remove the outliers, we thus search for the non-empty leaves of a BOT with all corners assigned the same tag and mark the enclosed points as the outliers. As illustrated in Figure 3, the isolated outlier cells may be identified by the growing process. When the number of outlier cells increases, the growing process alone may not be able to resolve the tags of the outlier cells. However, the corners of an outlier cell will still be observed through various lines-of-sight passing through the outliers during the carving process.

Note that this simple outlier detection procedure works under the assumption that some corners of the inlier cells are occluded by the inliers. Although this assumption is not guaranteed to be valid all the times, it is quite effective as shown in Section 5.1 and works for a large variety of real-world data sets if partitioned sufficiently.

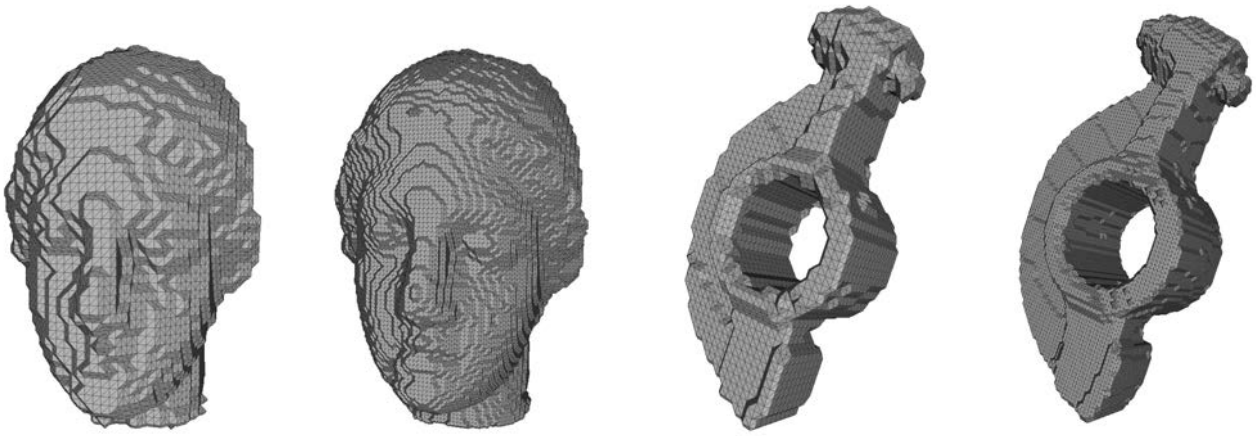
4. VOLUME AND SURFACE RECONSTRUCTION

To understand the inside/outside information with respect to a digital model is essential for many geometric modeling and processing problems. Note that the **in**/**out** tags carried by a BOT are intrinsically the same as the traditional volumetric representations and can thus be accepted by many existing iso-contouring methods such as the well-known *Marching Cubes* algorithm [16]. Because the BOT carries the input point cloud $\mathcal{P}$, it is adaptively refinable and can be converted into other volumetric representations, such as the *hermite data* [11], by embedding other information like intersection and the corresponding normal direction derived from $\mathcal{P}$. Due to the self-orientable property of mono-oriented cells, it is only necessary to tag the bi-oriented regions when being refined. In addition, because the tags of the mono-oriented cells at finer levels are implicitly recorded by their maximal mono-oriented parent, a BOT is a more compact representation than regular grids. Figure 4 illustrates several examples of volume reconstruction by using the BOTs.

There are many possibilities to extend the existing surface reconstruction algorithms to deal with unorganized point clouds with the aid of the BOTs. For example, the cell corners tagged as **in** or **out** themselves can serve as or may be used as a hint to create the *off-surface constraints* when reconstructing the *variational implicit surfaces* interpolating a given point cloud [21]. In this paper, we chose to extend the MPU implicits algorithm [17] for reconstructing surfaces with the BOTs by relaxing the requirement of normal vectors when reconstructing the surface from point cloud.

The MPU implicits algorithm is an octree-based local approximation method that computes a local shape function Q_i for each cell $\mathcal{C}_i$ by fitting a general quadric or bivariate quadratic polynomial to the local point set $\mathcal{P}_i$ contained by $\mathcal{C}_i$. To determine the orientation of Q_i, the corners $\mathbf{q}_i$ of $\mathcal{C}_i$ are used as auxiliary points and a general quadric is computed by minimizing

$$\frac{1}{\sum \omega(\mathbf{p}_i)} \sum_{\mathbf{p}_i \in \mathcal{P}_i} \omega(\mathbf{p}_i) Q_i(\mathbf{p}_i)^2 + \frac{1}{m} \sum_{i=1}^{m} (Q_i(\mathbf{q}_i) - d_i)^2, \quad (1)$$



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

Figure 4: The isosurfaces extracted from the BOT volumetric representation by the Marching Cubes algorithm [16]. (a) and (b) meshes consisting of 19,804 and 79,460 triangles extracted from BOT level 6 ($64\times64\times64$ grids) and 7 ($128\times128\times128$ grids) of the Igea data set, respectively. (c) and (d) meshes consisting of 10,024 and 40,584 triangles extracted from BOT level 6 ($64\times64\times64$ grids) and 7 ($128\times128\times128$ grids) of the RockerArm data set, respectively.

where ω is a weight function and d_i is the average signed distance computed from the nearest six points $\mathbf{p}_i \in \mathcal{P}_i$ of $\mathbf{q}_i$ by using normal vectors $\mathbf{n}_i$. Note that the **in**/**out** tags of $\mathbf{q}_i$ is provided by the BOTs, the signed distance d_i can be estimated similarly without $\mathbf{n}_i$.

Originally, a bivariate quadratic polynomial is defined in a local coordinate system (u, v, w) with the origin at the cell center $\mathbf{c}$ and the positive direction of w coincides with the direction of $\mathbf{n}$, which is the average normal vector of $\mathbf{n}_i$. Without using the normal vectors, we fit a bivariate quadratic polynomial by performing local covariance analysis on $\mathcal{P}_i$ to find out the best fitted plane as the local coordinate system and correct the orientation of the plane normal $\mathbf{n}'$ by maximizing

$$\sum_{i=1}^{m} sign(\mathbf{q}_i) \cdot (\mathbf{q}_i - \mathbf{c}) \cdot \mathbf{n}', \tag{2}$$

where $sign(\mathbf{q}_i)$ is a binary function that returns 1 or -1 if $\mathbf{q}_i$ is tagged as $+$ or $-$, respectively. Note that this simple method can also be applied to obtain the *globally consistent orientation* of an unsigned normal field, which was typically accomplished by using the orientation propagation methods based on traversing a corresponding minimal spanning tree [5, 7, 8, 23].

5. EXPERIMENTAL RESULTS

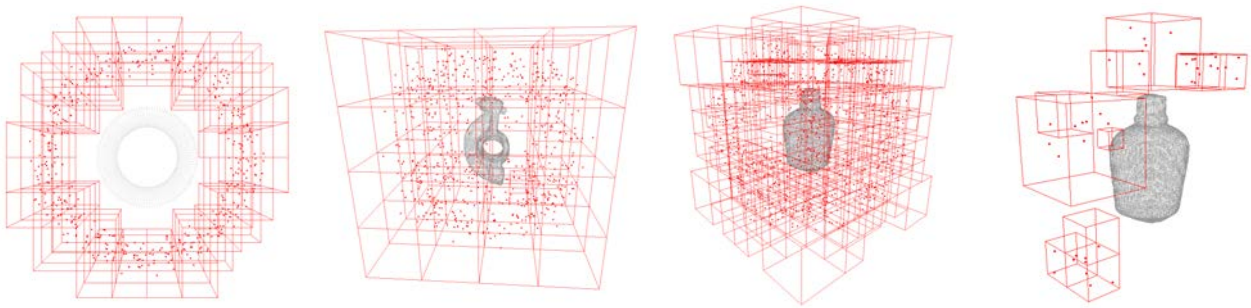
The HPR operator was implemented by using the Qhull algorithm [1] for convex hull computation. The complexity of tagging a BOT does not greatly increase with data sizes since a subset of a dense data set $\mathcal{P}$ is sufficient to hide the BOT corners inside $\mathcal{P}$ when performing the visibility check. Therefore, we take advantage of the BOT to compute a *particle* for each non-empty cell $\mathcal{C}_i$, which averages the points $\mathcal{P}_i$ in $\mathcal{C}_i$. Empirically, we used the particles at level 7 or 8 of the BOTs for the tagging process. Experiments are conducted by using some real-world data sets obtained from 3D scanning or image-based 3D reconstruction from image sequences, such as the Dinosaur. Note that a sparse data set like the Torus (4,800 points) is also included in our experiments.

Table 1: Statistics of reconstructing MPU implicit surfaces based on BOT from several data sets. The computation times for Tagging and MPU are represented in seconds.

Data Sets	Point #	Particles #	Tagging	MPU
Torus	4,800	3,591	0.188	1.844
Dinosaur	36,988	22,301	2.891	1.985
Rabbit	67,038	17,408	1.391	3.406
Santa	75,781	17,572	1.765	5.266
Igea	134,345	32,940	2.547	6.828

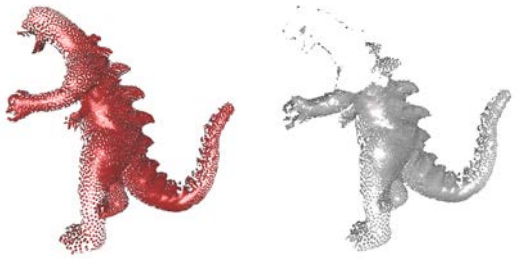
5.1 Outlier Removal

Figure 5 demonstrates the capability of the outlier removal during the construction of the BOTs. Randomly generated outliers are appended to the input point clouds for the BOT construction. Generally, all of the outliers can be effectively detected. In Figure 5 (c), some outliers were not detected because the increasing number of outliers caused some nearby cell corners to be hidden during the tagging process. After removing the detected outliers, such hidden corners are disoccluded and the remaining outliers can be detected by performing the tagging process again. It is interesting to note that Xie *et al.* [22] exploited a similar idea that outliers will be enveloped by mono-oriented regions during the growing of the active contour models for outlier detection. However, their method can only deal with a small number of outliers. In the cases shown in Figure 5, it is very likely that the outliers widely spread around will prevent the active contour models from reaching the real



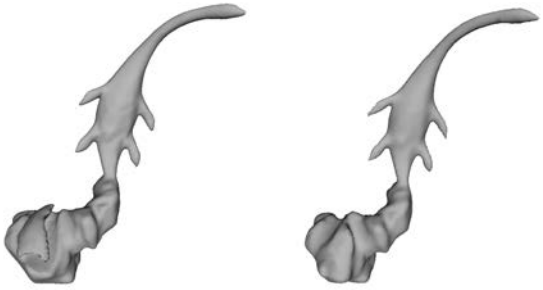
(a) (b) (c) (d)

Figure 5: The outlier removal results during the construction of the BOTs. The red cubes are the cells containing the detected outliers. (a) Torus data set (4,600 points with 500 outliers). (b) RockerArm data set (30,272 points with 700 outliers). (c) and (d) Bottle data set (35,086 points with 800 outliers), where 759 outliers are removed in (c) and the rest are detected by the second round of the BOT tagging process.



(a) (b)

Figure 6: Globally consistent normal estimation by BOT and orientation propagation [8]. The point clouds are rendered with the normal fields aligned by (a) BOT and (b) [8].



(a) (b)

Figure 7: Poisson surface reconstruction [13] of the Ness data set with the normal fields obtained by (a) [8] and (b) BOT.

data points.

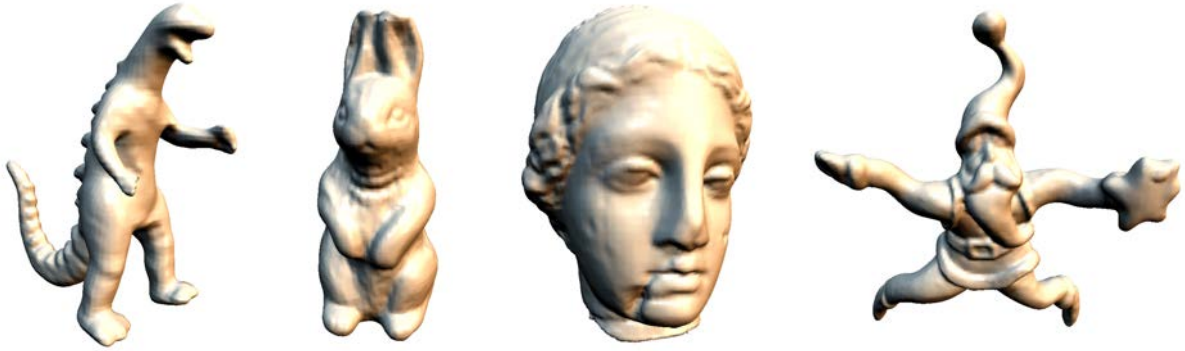
5.2 Surface Reconstruction and Orientation Determination

The modified MPU implicits algorithm explained in Section 4 was applied to several data sets and Table 1 summarizes the computation times measured on a desktop PC equipped with an Intel Core 2 2.93GHz CPU and 2GB main memory. Figure 8 demonstrates the MPU implicit surfaces reconstructed by using the BOTs. One can see that the BOTs maintain the efficiency of the MPU implicits algorithm while enabling it to deal with unoriented data sets.

We have also exploited the BOTs to align an unsigned normal field estimated from $\mathcal{P}$ with each normal vector ambiguously directed inward or outward. For comparison, we chose the state-of-the-art orientation propagation algorithm [8]. As shown in Figures 6 and 9, the point sets are rendered with the aligned normal fields by splating. With back culling enabled, one can observe that some data points become invisible due to the erroneously directed normals obtained by [8], while the results by BOTs do not have such problems. If the data points with erroneous normal vectors are passed into other reconstruction methods like Poisson surface reconstruction [13], it will certainly produce incorrect results, as shown in Figure 7 (a). Though being generally effective, the orientation propagation methods [5, 7, 8, 23] work on propagating local information to obtain a global property. However, the orientation of nearby normal vectors can actually vary drastically due to sparsity, non-uniformity or sharp features presented in $\mathcal{P}$. In contrast, the **in**/**out** information indicated by BOTs is more globally compliant with the true orientation of $\mathcal{P}$. In spite of its simplicity, it is thus more robust to orientate the individual normal vectors by the nearest oriented corners of a BOT.

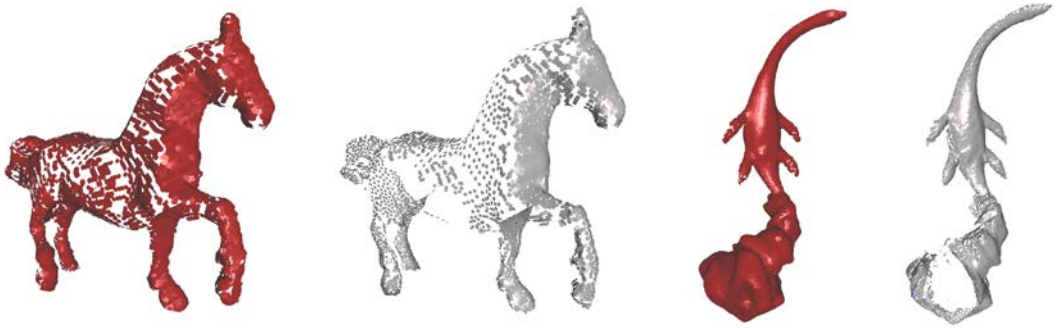
6. CONCLUSION

In this paper, we presented the binary orientation trees (BOTs) which are built from unoriented point clouds by octree subdivision and associated with the orientation information derived through performing the visibility checks from various viewpoints. Being conceptually simple, BOTs are easy to implement and computationally efficient to compute. BOTs resemble the traditional volumetric data representations and are advantageous to applications, such as surface reconstruction and orientation determination. Currently, BOTs are limited to complete data sets. For the future work, we plan to utilize the data points carried by BOTs to derive more useful metadata for volume reconstruction and also guide the inside/outside space partitioning which



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

Figure 8: MPU implicit surfaces based on the BOTs, which are reconstructed from the (a) Dinosaur, (b) Rabbit, (c) Igea and (d) Santa data sets.



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

Figure 9: More examples of globally consistent normal estimation by BOT and orientation propagation [8] obtained from the Horse (18,532 points) and Ness (25,798 points) data sets. (a) and (c) results by BOT. (b) and (d) results by [8].

cannot be fully resolved by visibility checks.

7. REFERENCES

[1] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, 1996.

[2] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[3] T. Boubekeur, W. Heidrich, X. Granier, and C. Schlick. Volume-surface trees. *Computer Graphics Forum*, 25(3):399–406, 2006. (Eurographics 2006 Conference Proceedings).

[4] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. In *ACM SIGGRAPH 1980 Conference Proceedings*, pages 124–133, 1980.

[5] G. Guennebaud and M. Gross. Algebraic point set surfaces. *ACM Transactions on Graphics*, 26(23):23, 2007. (SIGGRAPH 2007 Conference Proceedings).

[6] C.-C. Ho, F.-C. Wu, B.-Y. Chen, Y.-Y. Chuang, and M. Ouhyoung. Cubical marching squares: Adaptive feature preserving surface extraction from volume data. *Computer Graphics Forum*, 24(3):537–545, 2005.

[7] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *ACM SIGGRAPH 1992 Conference Proceedings*, pages 71–78, 1992.

[8] H. Huang, D. Li, H. Zhang, U. Ascher, and D. Cohen-Or. Consolidation of unorganized point clouds for surface reconstruction. *ACM Transactions on Graphics*, 28(5):176, 2009. (SIGGRAPH Asia 2009 Conference Proceedings).

[9] C. L. Jackins and S. L. Tanimoto. Oct-trees and their use in representing three-dimensional objects. *Computer Graphics and Image Processing*, 14(3):249–270, 1980.

[10] A. C. Jalba and J. B. T. M. Roerdink. Efficient surface reconstruction from noisy data using regularized membrane potentials. *IEEE Transactions on Image Processing*, 18(5):1119–1134, 2009.

[11] T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual contouring of hermite data. *ACM Transactions on Graphics*, 21(3):339–346, 2002.

[12] S. Katz, A. Tal, and R. Basri. Direct visibility of point sets. *ACM Transactions on Graphics*, 26(3):24, 2007. (SIGGRAPH 2007 Conference Proceedings).

[13] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Proceedings of the 2006 Eurographics Symposium on Geometry Processing*, pages 61–70, 2006.

[14] M. Kazhdan, A. Klein, K. Dalal, and H. Hoppe. Unconstrained isosurface extraction on arbitrary octrees. In *Proceedings of the 2007 Eurographics Symposium on Geometry Processing*, pages 125–133, 2007.

[15] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel. Feature sensitive surface extraction from volume data. In *ACM SIGGRAPH 2001 Conference Proceedings*, pages 57–66, 2001.

[16] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM SIGGRAPH Computer Graphics*, 21(4):163–169, 1987. (SIGGRAPH 1987 Conference Proceedings).

[17] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel. Multi-level partition of unity implicits. *ACM Transactions on Graphics*, 22(3):463–470, 2003. (SIGGRAPH 2003 Conference Proceedings).

[18] M. Pauly, M. Gross, and L. P. Kobbelt. Efficient simplification of point-sampled surfaces. In *IEEE Visualization 2002 Conference Proceedings*, pages 163–170, 2002.

[19] H. Samet. *Quadtree, Octrees, and Other Hierarchical Methods*. Addison Wesley, 1989.

[20] I. Tobor, P. Reuter, and C. Schlick. Multi-scale reconstruction of implicit surfaces with attributes from large unorganized point sets. In *Proceedings of the 2004 Shape Modeling International*, pages 19–30, 2004.

[21] G. Turk and J. F. O'brien. Modelling with implicit surfaces that interpolate. *ACM Transactions on Graphics*, 21(4):855–873, 2002.

[22] H. Xie, K. T. McDonnel, and H. Qin. Surface reconstruction of noisy and defective data sets. In *IEEE Visualization 2004 Conference Proceedings*, pages 259–266, 2004.

[23] H. Xie, J. Wang, J. Hua, H. Qin, and A. Kaufman. Piecewise c1 continuous surface reconstruction of noisy point clouds via local implicit quadric regression. In *IEEE Visualization 2003 Conference Proceedings*, pages 91–98, 2003.

[24] H.-K. Zhao, S. Osher, and R. Fedkiw. Fast surface reconstruction using the level set method. In *Proceedings of the 2001 IEEE Workshop on Variational and Level Set Methods*, pages 194–201, 2001.