Proceedings of DETC'03
ASME 2003 Design Engineering Technical Conferences and
Computers and Information in Engineering Conference
Chicago, Illinois USA, September 2-6, 2003

DETC2003/DAC-48779

# PARTITIONING POSITIONAL AND NORMAL SPACE FOR FAST OCCLUSION DETECTION

**Xiaoping Qian**
qian@crd.ge.com
Inspection & Manufacturing Technologies
GE Global Research
P.O. Box 8, Niskayuna, NY 12309

**Kevin G. Harding**
harding@crd.ge.com
Inspection & Manufacturing Technologies
GE Global Research
P.O. Box 8, Niskayuna, NY 12309

**ABSTRACT**

Occlusion detection is a fundamental and important problem in optical sensor inspection planning. Many view-planning algorithms have been developed for optical inspection, however, few of them explicitly develop practical algorithms for occlusion detection. This paper presents a hierarchical space partition approach that divides both positional and surface normal space of an object for fast occlusion detection. A k-d tree is used to represent this partition. A novel concept of $\delta$ – *occlusion* is introduced to detect occlusion for objects in an un-organized point cloud representation. Based on the $\delta$ – *occlusion* concept, several propositions regarding to a range search on a k-d tree have been developed for occlusion detection. Implementation of this approach demonstrated that significant time can be saved for occlusion detection using the partition of both positional and surface normal space.

**Keywords:** Visibility, Occlusion Detection, Spherical Map, K-d Tree

## 1 introduction

Optical sensors have been used in many 3D shape measurement applications such as optical metrology and reverse engineering. Effective and efficient inspection planning of these sensor systems is a critical task in optical applications. Many planning systems have been developed for this purpose.

The task of inspection planning is to seek a set of viewpoints for the sensor so that all the points on the part can be inspected optimally from these viewpoints. It involves the determination of extrinsic and intrinsic parameters for the sensors, optical illumination parameters for the lasers, and optimal placement of lasers and sensors so that part surfaces can be inspected without occlusion or collision. In this paper, we focus on one aspect of inspection planning: developing fast algorithms to ensure occlusion-free inspection.

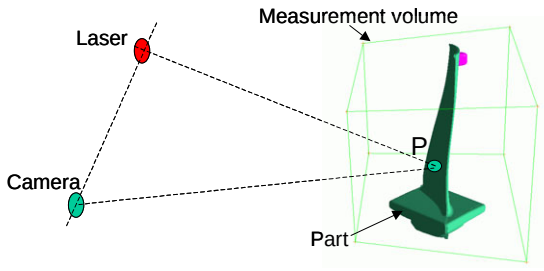



**Figure 1 A valid inspection point for an optical inspection system**

To ensure a point on an object is inspectable, this point must satisfy certain requirements (Figure 1). They include *1) the point must lie within the measurement volume MV, formed by the field of view (FOV) and the depth of field (DOF); 2) the point must be visible to both the laser and the camera, 3) the point must be accessible, that is, there should be no occlusion between the point and the laser or between the point and the camera.*

The second condition is often referred to as a visibility constraint. In this paper, we refer to a point as *locally visible* when the angle of incidence of this point (angle between the surface normal of a point and the line-of-sight formed by the point and the laser or camera) is smaller than 90 degrees. We refer to a point as *globally visible* when 1) the angle of incidence of this point is smaller than 90 degrees, and 2) no point in the object occludes this point. From a given set of

points that are locally visible to both a laser and a camera, we need to find out those points that are not occluded by any object. In other words, we need to find all the points that are globally visible. These are the points that can be inspected under a given viewpoint.

In a typical sensor inspection planning system, candidate views are generated first. A model coverage test is then conducted to ensure the object is fully covered. Often times, the occlusion check is not explicitly explored. Thus, some locally visible points may not be inspected due to occlusion.

This paper presents an efficient approach for fast occlusion check based on space partition. Position space partition is often used for collision detection. However, the occlusion detection involves the surface normal check. The approach in our paper is based on a hierarchical partitioning of both position space and surface normal space of an object. The object to be inspected is represented in a tessellated model with each point having a position coordinate and a surface normal.

We transform the occlusion problem using the $\delta$ – *occlusion* concept, so that we can use a k-d tree based space partition for occlusion detection. For a constructed k-D tree, we use the line-of-sight to intersect the 3d bucket. A k-d tree of six dimensions, consisting of the position value and the surface normal of each point, is constructed for the object. For each candidate view point, the problem of finding the points from the object falling into the measurement volume and satisfying the visibility constraints is transformed into a range search problem in a k-d tree.

In the remaining of this paper, Section 2 reviews prior work on inspection planning and occlusion detection. Section 3 details our approach, including $\delta$ – *occlusion,* a simple algorithm for point inspectability check and an improved algorithm based on space partition. Section 4 presents the implementation of the algorithms and computational results. The paper is concluded in Section 5.

## 2 Literature Review

Many sensor planning systems have been developed for the best placement of sensors and parts to ensure full model coverage, to enhance inspection quality and to decrease build time. The planning methods behind these systems largely fall into two categories: 1) a generate-and-test approach, and 2) a synthesis approach (Tarabanis 1995). In a generate-and-test approach, the sensor configurations are generated and then evaluated with respect to task objectives. In order to limit the number of candidate sensor configurations, the configuration space is discretized. For example, an approach using known imaging sensors and feature-based object model to compute the optimal positions for inspection tasks is reported in (Trucco 1997). In the synthesis approach, the task requirements are characterized analytically and the sensor parameter values that satisfy the task constraints are directly determined from these analytical relationships. For example, a computational approach for best sensor setup to minimize signal dynamic range is reported in (Qian 2003).

In all these inspection planning systems, computing occlusion-free view points is a complex task and consumes a lot of computing time. Most of the inspection planning systems do not provide efficient methods for occlusion detection. For example, a general planning system is developed for laser scanning, in which occlusion check is done for every view points in the scan plan, but with no explicit mentioning of the algorithm's efficiency (Son 2002). A cone visibility based method is used to determine a near optimal decomposition of free form surfaces according to surface normal (Elber 1998). In the context of NC machining, spherical geometry based visibility has been extensively explored (Chou 1992). However, in these approaches, only local visibility is addressed and global visibility is not addressed.

Methods to calculate the occlusion-free loci in terms of boundary representation and CSG representation were developed in (Tarabanis 1996). Exact aspect graphs have also been used to analyze CAD models to obtain exact and continuous visibility regions covering the whole space (Petitjean, 1990). However, these methods are rarely applied in practice, in part because of their computational and representational complexity (Trucco 1997).

A concept similar to occlusion detection is collision detection. If the geometry of two objects is intersecting with each other, there is collision between these two objects. Many algorithms have been developed for efficient collision detection. Fast collision detection algorithms typically employ a divide-and-conquer scheme by sub-dividing the object's 3D space (Lin 1998). For example, a tessellated representation with a hierarchy space division is developed for collision detection in (Gottschalk 1996). In this approach, they used a Discrete Orientation Polytope (DOP). A hierarchical adaptive space subdivision scheme, the BoxTree, and an associated divide-and-conquer traversal algorithm were developed for interactive virtual prototyping (Zachmann 1997). Typically hierarchical schemes outperform the non-hierarchical schemes as long as the hierarchies are not required to re-build dynamically.

Occlusion check differs from collision check in that occlusion can happen even when there is no collision. Occlusion happens when line-of-sight from a source point is intersecting with any object geometry before it hits the target point. In collision detection, typically only position space partition is required. In the paper, we explore the partition of both positional and surface normal space partition for fast occlusion detection.

In this paper, we use a k-D tree (k-dimensional binary-search tree) based representation for partitioning both positional and surface normal space. A k-D tree is a common data structure used to find a point's closest neighbor in a point set (Friedman 1977, Preparata 1988). A k-D tree partitions space into a set of buckets. A hyper-sphere is then used to intersect the bucket to find the closest neighbor. The computational complexity is O(nlogn) for constructing a k-D tree, and O(logn) for searching such a tree.

## 3 Overview of The Fast Occlusion Detection Approach

In this section, we first introduce the concept of $\delta$ – *occlusion.* We then present a simple algorithm for point inspectability check and an improved algorithm based on space partition.

## 3.1 Proposition for point cloud based occlusion test

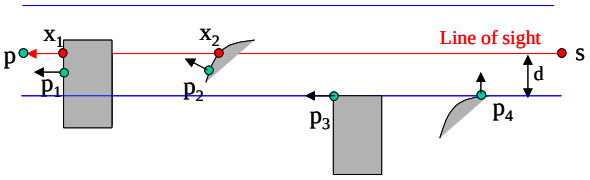



**Figure 2 Assumptions about the object and the tesselation**

When a line-of-sight intersects with an object, there would be at least two intersection points. One of the intersection points would be invisible to the source point. For example, in Figure 2, the line-of-sight $\overrightarrow{sp}$ intersects with the objects at the point $x_1$ and the point $x_2$, both x1 and x2 are invisible to sensors. Therefore, we have the following lemma for the occlusion test.

Lemma 1: *If among all the points in the point cloud of an object, if we can find one point x, which meets the following conditions: 1) it is on the line-of-sight, and 2) its surface normal forms an angle with incidence vector* $\overrightarrow{sp}$ *smaller than 90, we then know that point p is occluded and point x is an occluding point*.

The line-of-sight in theory has an infinite-small diameter. The direction application of an intersection test between a line-of-sight and a point cloud does not always produce the occluding points. In order to conduct the occlusion test for the point cloud representation of an object, we make the following assumptions about the point cloud representation of an object:

1) distance between each point no larger than δ,
2) minimal feature size δ,
3) curvature radius > $\frac{2\delta}{\pi}$ .

These assumptions ensure that, if there is any intersection between a line-of-sight and an object and the intersections points are within the δ distance from the line-of-sight, at least one point invisible to the sensor will be in the point cloud.

Based on these assumptions, we have the following proposition for occlusion check for point cloud.

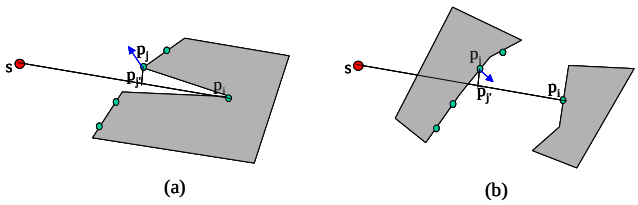



**Figure 3 Occlusion test**

**Proposition 1** *δ – occlusion*

Let sensor point *s* and candidate point $p_i$ from an object point set P form a line-of-sight $\overrightarrow{sp}_i$. For any point $p_j$ from the object point set P, let the distance between the point $p_j$ and the line-of-sight be *d*, and let the $p_j'$ be the closest point from the $\overrightarrow{sp}_i$ to $p_j$. Let the angle between the surface normal $\overrightarrow{n}_i$ at point $p_i$ and the line-of-sight to be θ. That is, $\theta = a\cos(\overrightarrow{n}_i \cdot \frac{\overrightarrow{sp}_i}{\|sp_i\|})$.

If the following conditions are met,

1) $d < \delta$,
2) $\overrightarrow{p_j'p_i} \cdot \overrightarrow{p_j's} < 0$,
3) $\theta <= 90$,

then candidate point $p_i$ is either occluded by the surface patch around the point $p_j$ or at most δ distance away from being occluded by the surface patch. We refer to this "occlusion" as *δ – occlusion*.

**Proof:**

According to the 1st condition, we know point $p_j$ is within δ distance from the line-of-sight. From the 3rd condition, we know the point $p_j$ is invisible to the sensor *s*. From the 2nd condition, we know the shortest point between the line segment $\overrightarrow{sp}_i$ and the point $p_j$ lies within the line segment. If we move the line-of-sight $\overrightarrow{sp}_i$ by d distance, so that point $p_j$ lies on top of $\overrightarrow{sp}_i$, then we can know the moved line-of-sight is occluded by point $p_j$. That is, the point $p_j$ is at most *d* distance away from being occluded by the surface patch around the point $p_j$. Due to the sample rate of *δ* distance, point $p_i$ is occluded by the surface patch or at most *δ* distance away from being occluded by the patch.

From the above proposition, we know for a given point $p_i$, if any one point satisfies the conditions, then the point $p_i$ is *δ–occluded.* If no point from the object set meets the requirements, then the point $p_i$ is occlusion-free. That is, point $p_i$ is globally visible and accessible for optical inspection. So the complexity of occlusion test for any given candidate point is *O(n)* where *n* is the number of points in the point set.

Note that the *δ–occlusion* may overshoot the occlusion test by distance δ. However, typically δ can be made very small when the object is sampled into a point cloud. Due to the manufacturing tolerance and the inaccuracy during the part setup, it is often expected the inspection point to be some distance away from being occluded.

## 3.2 Simple algorithm for part inspectability test

A part is fully inspectable when all the points on the part can be inspected. In order to check if all the points can be inspected for a given inspection plan, all the points need to be checked against the inspectability requirements. That is, each point has to 1) lie within measurement volume, 2) be locally visible to both the laser and the camera, 3) be accessible or occlusion-free.

A simple algorithm for part inspectability check is as follows (Algorithm 1).

**ALGORITHM 1: A SIMPLE ALGORITHM FOR PART INSPECTABILITY CHECK**

For each view point $V_i$ in the view plan

1.0 Obtain the measurement volume $MV_i$ from the view point $V_i$

2.0 For each point $p_j$ from the part P

If $p_j \in MV_i$, then add $p_j$ to the "within" point set $W_1$.

Endfor

3.0 For each point $p_j$ within the measurement volume, $p_j \in W_1$

If point $p_j$ is visible to both the laser and camera, then add $p_j$ to the local visible set $W_2$.

Endfor

4.0 For each locally visible point $p_j$, $p_j \in W_2$

If point $p_j$ is occlusion free, then mark this point as inspectable.

Endfor

Endfor

5. 0 For each point $p_j$ from the part P

If $p_j$ is not marked inspectable, then add $p_j$ to the un-inspected point set U.

Endfor

The input is a set of view plans. The output is a point set *U* that contains all the points not covered by any viewpoint. It involves the following five steps.

**1. Obtain the measurement volume for each view point.**

**2. Find all the points within the measurement volume.**

For a given view point, the measurement volume is typically a trapezoid volume (Figure 1) and it can be represented as a collection of six planar half-spaces $MV = \{(p - p_i) \cdot n_i <= 0 \mid i = 1,6\}$, where each plane is represented by a point $p_i$ and a plane normal $n_i$. So whether a point lies within the measurement volume can be determined through a set of half-space calculations.

3**. Check if all the "within" points are locally visible to both the laser and the camera.**

The local visibility can be determined through the surface normal comparison. For a candidate point p with surface normal $\vec{n}$, if $\vec{n} \cdot (s - p) > 0$, where s is either the laser position or the camera position, then point p is visible to the sensor s.

**4. For all the points that are locally visible, we check if they are occlusion free.**

The occlusion test is based on the δ-occlusion proposition in the last section. For a given sensor (either laser or camera) position *S* and a candidate point $p_j$, we have this occlusion check algorithm. The output of this algorithm is whether the candidate point $p_j$ is occluded.

**Algorithm 2: A Simple Algorithm for Occlusion Check**

For each point $p_i$ from the part P

Project point $p_i$ onto the line $p_j s$ and obtain the projected point $p_i'$

Calculate distance d between $p_i$ and $p_i'$, $d = \|p_i p_i'\|$

Calculate the angle θ between $p_j s$ *and surface normal* $\vec{n}_i$,

$$\theta = a\cos(\vec{n}_i \cdot \frac{\overrightarrow{sp_i}}{\|sp_i\|})$$

If $d < \delta$, $\overrightarrow{p_j' p_i} \cdot \overrightarrow{p_j' s} < 0$, and $\theta <= 90$

Return True

Endfor

Return False.

**5. Check if any point is not covered by all the viewpoints.**

Algorithm 2 has complexity of *O(n)* where *n* is the number of points in the point cloud of the part. So the Algorithm 1 (Simple algorithm for part inspectability test) has complexity of $O(n^2)$.

## 3.3 Space partition based algorithm for part inspectability check

We improve the simple algorithm in last section using space partition via a k-d tree. We first present the construction of a k-d tree based on positional space and normal space partition. We then transform the simple algorithm into a range search problem on the k-d tree.

### 3.3.1 Partitioning position and normal space

For many spatial problems, space partition is a common scheme to increase computational efficiency. Since the inspectability problem involves both position and surface normal constraints, we propose the simultaneous partition of positional and normal space to increase the algorithm efficiency.

### Construction of a k-d tree

We adopt a classical k-d tree construction method (Preparata 1988) for our problem. In this method, a boundary box is needed for space partition. Different types of boundary boxes have been used for a variety of applications. For the simplicity of implementation, in this paper, we use axis-aligned boundary box. However, the algorithms presented in this paper are applicable to other types of boundary boxes.

For each point, three are three positional values *(x, y, z)* and three surface normal values *(nx, ny, nz).* We represent the six values as a 6-dimensional point *(x, y, z, nx, ny, nz).* Suppose that all the points from an object to be inspected lie in a 6-dimensional box, which has some points identified on its boundary. We build the data structure inside this box, and define it recursively as follows. The set of data points is split into two parts by splitting the box containing them into two child boxes by a hyper plane, according to some splitting rule specified by the algorithm; one subset contains the points in one child box, and another subset contains the rest of the points. The information about the splitting plane and the boundary values of the initial box are stored in the root node, and the two subsets are stored recursively in the two subtrees. When the number of the data points contained in some box falls below a given threshold, then we call a node associated with this box a leaf node, and we store a list of coordinates for these data points in this node. (Preparata 1988).

In this paper, we split the boundary box sequentially in x, y, z, nx, ny, and nz order. When the split happens in the positional space, the surface normal boundary of the set may be changed. Likewise, when the split happens in the surface normal space, the boundary box of the positional space may also change. In order to have a tight boundary box, during the

construction process, we calculate and record the new boundary box after each split.

## Range search in a k-d tree

The range search of a k-d tree is handled differently from a standard k-d tree. We have the following range search algorithm (Algorithm 3). The output is a set of points, *W*, falling within the range R. The algorithm works recursively. First, it checks if the node is a leaf. If it is a leaf, it finds out all the points under the leaf falling into the range *R* and adds then into *W*. If it is not a leaf, it checks if the boundary box of the node, *extent*, completely lies within the range *R*. If it does, all the points under the node fall into the range *R*. It then checks if the boundary box overlaps with the range. If it does, it recursively searches its left sub-tree and right sub-tree. If not, it returns the point set *W*.

**Algorithm 3 Range Search based on a k-d tree**

```
RangeSearch(node N, BoundaryBox Extent, Range R)
If (type(N) == leaf)
{
        For each point pi in the leaf node
                If (isWithin(pi, R) == true), then W ⇐ p_i
        Endfor
}
Else
{
        Axis = node->SplitAxis;

        If(isWithin(Extent, R) == true)
                W ⇐ all the points under N
        If(isOverlap(Extent, R) == true)
        {
                // Left Child
                TmpExtent = Extent[2*Axis];
                Extent[2*Axis] =  N->SplitValue;
                RangeSearch( N->LeftChild, Extent, R);
                Extent[2*Axis] = TmpExtent;

                // Right Child
                TmpExtent = Extent[2*Axis];
                Extent[2*Axis] =  N->SplitValue;
                RangeSearch( N->RightChild, Extent, R);
                Extent[2*Axis] = TmpExtent;
        }
}
```

In the above range search algorithm on a k-d tree, there are three basic Boolean interaction tests: 1) whether a point $p_i$ lie within a range *R*, isWithin($p_i$, R), 2) whether a boundary box *extent* falls completely within the range *R*, isWithin(Extent, R), 3) whether the boundary box *extent* overlaps with the range *R*, isOverlap(Extent, R). The first test is a basic containment test. The second and the third tests enable the search time saving on a hierarchal tree. For example, in Figure 4, a point set is shown in the left figure and its space partition is shown in the right figure. In order to find all the points within the range A, the k-d tree will be selectively traversed. In this particular case, only nodes 6, 3, 2, and 5 will be checked for Range A. This information can be obtained through the overlap test isOverlap(Extent, R) since the boundary boxes of these nodes overlap with the Range A. In order to search the points in Range B, the nodes 6, 3, 2, 1, will be checked. Due to the fact that Range B falls completely within the boundary box of node 1's left child, any points in the left leaf of node 1 is automatically within the Range B. This information is obtained through the isWithin(Extent, R) test.

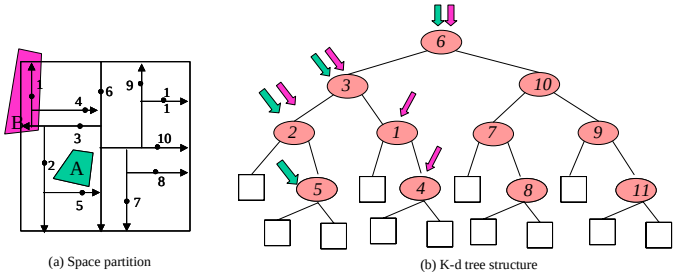



**Figure 4 Interaction tests between boundary box and range**

### 3.3.2 Convert a point inspectability test into a range search problem

In order to take advantage of the computational efficiency of a range search based on hierarchal positional and normal space partition, we convert the point inspectablity test into a range search problem on the k-d tree. We then present several propositions related to a range search for occlusion detection.

We divide the point inspectability problem into a two-step problem. During the first step, we seek all the points that lie within the measurement volume and are visible to both the camera and the laser. We refer to these points as locally inspectable points, meaning these points are inspectable in a local sense, without knowledge whether they meet the global visibility conditions or not. During the second step, we check if any point in the locally inspectable point set meets the global visibility conditions, i.e. whether it is occluded.

**Range search for locally inspectable points**

The problem of finding points within the measurement volume, and meeting visibility criteria is essentially equivalent to a range search problem. The range is a combination of a measurement volume in the positional space and a visibility cone in the gaussian map (Qian 2003) in the surface normal space. The search time can be saved in two ways when a k-d tree is used. One is, when the bounding box does not overlap with the measurement volume or the visibility cone, there is no need for the inspectability test for the points within the measurement volume. None of these points can be inspected. The second is, when the bounding box and surface normal spherical convex hull are within the measurement volume and the visibility cone, there is no need for the test for the points within the measurement volume. All the points can be inspected. When there is overlap between a node's boundary box and measurement volume (or visibility cone), then we recursively conduct a range search for its left and right sub-trees.

In order to use the range search based on a k-d tree, we put forward the follow propositions with regarding to the interaction between the range (measurement volume and visibility cone) and the positional and normal space boundary box.

**Proposition 1**: If a positional boundary box does not overlap with measurement volume, or the boundary box of surface normal does not overlap with visibility cone of the sensor, none of the points within the boundary box are valid points for measurement. *(Completely Out)*

**Proposition 2**: If the bounding box is within the measurement volume, and all the eight surface normal of the normal boundary box are within the visibility cone, then all the points under this k-d tree node are locally effective measurement points. *(Completely Within)*

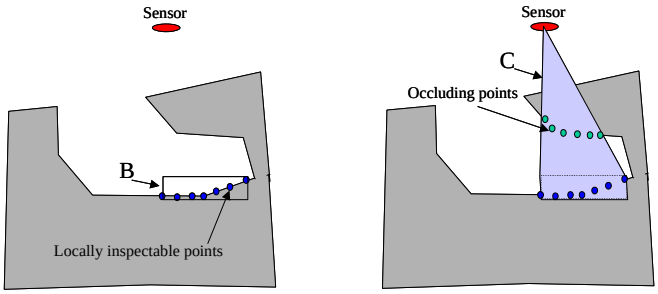
The two propositions respectively correspond to the range test, isOverlap(Extent, R) and isWithin(Extent, R), in the k-d tree. The first proposition reveals that the node does not overlap with the range. The second proposition shows that all the points in this node are completely within the range. If we substitute these two functions into Algorithm 3 based on the two propositions, we get an improved algorithm for finding the locally inspectable points.

**Range Search for Occlusion-free Points**

Once we have the locally visible points, we need to verify if each point is occlusion-free. The simple algorithm (Algorithm 2) compares each locally inspectable point against the whole point cloud and has the complexity of O(n). We improve this simple algorithm by comparing the locally inspectable points against the potentially occluding points.

The potentially occluding points are those points falling into the convex hull formed by the boundary box of the locally visible points and the sensor. As shown in Figure 5, for a set of locally inspectable points (left figure), we can form the boundary box *B*. We then form a convex hull *C* between the sensor and boundary box *B*. Any point falling into the convex hull other than the locally inspectable points, if this point is not visible to the sensor, is a potentially occluding point. Note, according to *δ-occlusion* proposition, we only use the point that is not visible to the sensor as a basis for occlusion detection. Therefore, the issue of finding potentially occluding points is again a range search problem. Here, the range is a combination of convex hull in the positional space and the visibility cone in the normal space. Substituting this range into Algorithm 3 leads to improved algorithm for finding potentially occluding points. In this improved algorithm, we use the following proposition for range search function isWithin(Extent, R).

**Proposition 3**: If there is no point within the convex hull formed by the boundary box of the locally inspectable points and the sensor, then all the points under this k-d tree node are globally effective measurement points, i.e. occlusion-free. *(Completely Within)*




(a) Find the boundary box of locally inspectable points

(a) Form the convex hull and find the potentially occluding points

**Figure 5 Seeking potentially occluding points**

Once we find potentially occluding points, we compare each locally inspectable point against the potentially occluding point set. Such a comparison is also a range search problem. In the positional space, the range is a cylinder. The cylinder's centerline is a line with end points at the locally inspectable point and the sensor. The radius is $\delta$. In the surface normal space, the range is the whole space. For each point within the cylinder, we then compare its normal to see if the angle between the normal and the line-of-sight (formed between the sensor and the point) is larger than 90 degrees (the second condition of Lemma 1).

### 3.3.3 Range and Bounding box interaction test

We convert the point inspectablity problem, including occlusion, into a set of range search problem on a k-d tree. It involves a set of geometric interaction test between the boundary boxes (of positional and normal space) and the ranges. The boundary boxes are axis-aligned boxes. In positional space, the ranges include measurement volumes for finding locally inspectable points, convex hulls for finding potentially occluding points, and cylinders for determining if a point is occluded by the potentially occluding point set. In surface normal space, the ranges include the visibility cones of a set of vertices in the boundary boxes of the surface normal.

Due to the page limit, we will not go into details about the interaction test between the boundary boxes and the ranges. The basic idea behind our interaction tests is using a dimension reduction method to convert a 3D geometry model and bounding box overlap test into a lower dimensional geometry interaction test. For example, we can convert a 3D cylinder and 3D boundary box interaction test into a problem of a point-line distance calculation and a line-plane intersection test. Specifically, we first check the extreme cases. They include: 1) All the bounding box vertices lie within the cylinder. This can be tested using point-line distance calculation. 2) The cylinder centerline lies within the bounding box. This can be easily known by checking if end points of the centerline lie within the box. 3) The two end points of the cylinder center lines lie at least r distance away from the boundary box. After the special cases processing, we intersect the centerline with the box. For each boundary plane of the box, we offset the centerline by distance of cylinder raidus and intersect the offset line with the plane. If the end points of both the centerline and the offset centerline lie outside of boundary plane, there is no overlap. If the intersection point of offset line with the boundary plane lies beyond the end points of the offset line, there is overlap. If either intersection point (from center line or from offset lines) lies within the boundary box, there is overlap. Otherwise, there is no overlap.

## 4 Implementation and results

The algorithms have been implemented on an HP-UX 11.0 U9000/785 machine. We compare the computational results of the algorithms based on one benchmark part (Figure 6). The tessellated model and its gaussian map are also included in the figure. The tessellated model includes 16553 points and surface normal.

A snapshot of the first 20 boundary boxes during the k-d tree construction process is shown in Figure 7 and Figure 8, respectively illustrating the partitioning of positional space and surface normal space. Figure 7 shows one picture in which the

tessellated model lies within the boundary box, and one picture without the tessellated model. Figure 8 shows a wire-frame and a shaded model of the surface normal space partition. Different colors represent the boundary box at different levels in the k-d tree. The time for constructing the k-d tree of 16553 points is about 0.355 second.

(a) Inspection part (b) Tessellated inspection surfaces (c) Surface normal of inspection surfaces

**Figure 6 A benchmark part**

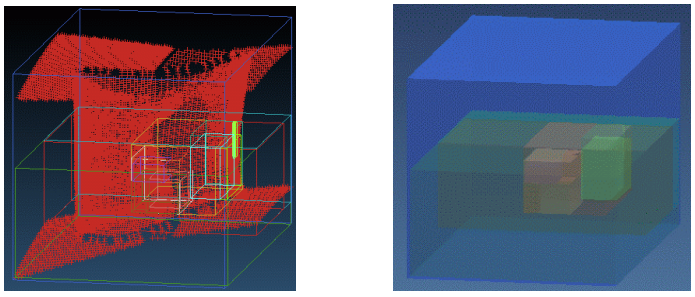


**Figure 7 Partitioning positional space**

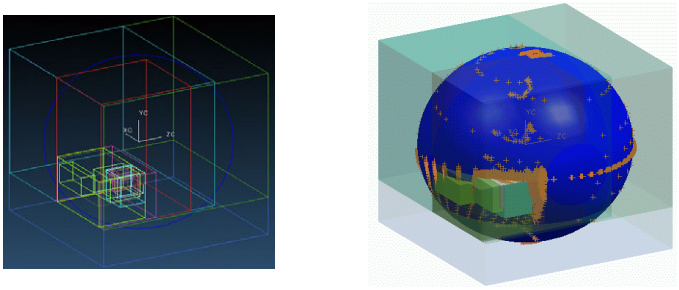


**Figure 8 Partitioning surface normal space**

In order to compare the computational time during point inspectablity check for different algorithms, we select seven inspection views, under which we calculate 1) the points that are within the measurement volumes and are locally visible, and 2) the points that are globally visible, i.e. no occlusion.

We first compare time spent in search of valid points within measurement volume and locally visible to both the camera and the laser using either partitioning positional space only or partitioning both positional and surface normal space. Figure 9 shows the time comparison of these two methods for different sensor view angles. As the sensor view angle constraints changes from 55 to 85 degrees, the method using partitioning both positional and surface normal space consistently outperformed the method using partitioning positional space only. Figure 10 shows the time comparison of these two methods for different field of views. The largest size of the field of view and depth of field is 24″*24″*18″. We then tried to set the field size to one half, one fourth, and one sixth of its original size. The respective results are shown in Figure 10. Again, the method using the partitioning of both positional and surface normal space consistently outperformed the method using the partitioning of positional space only regardless of the size of the measurement volume.

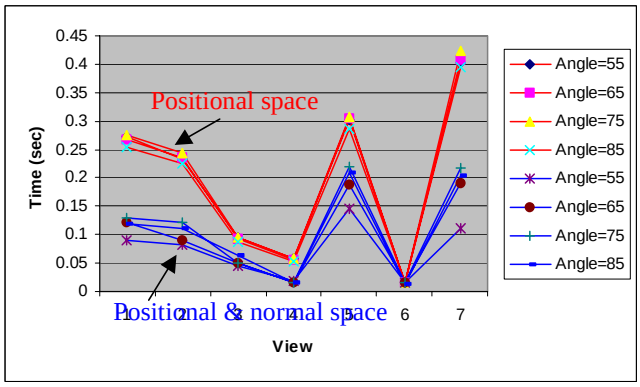
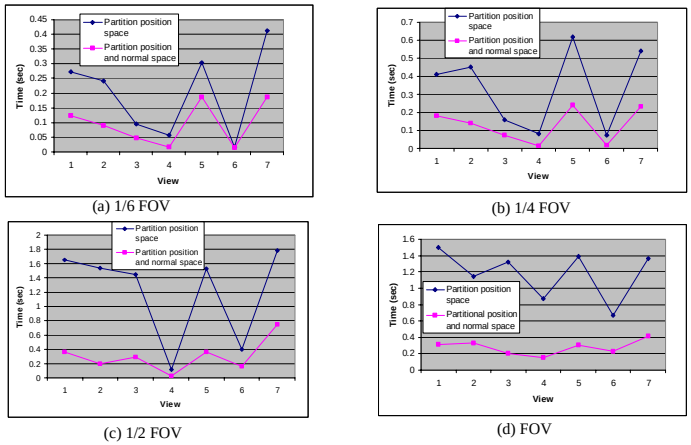



**Figure 9 Time comparison for different sensor view angles**




(a) 1/6 FOV (b) 1/4 FOV (c) 1/2 FOV (d) FOV

**Figure 10 Time comparison for different field of views**

Finally, we compare the time consumed in search of both locally visible points and the occlusion-free points for three methods under the same sensor view angle and the same measurement volume constraints. The results are shown in Table 1. As the table shows, the space partition based approach, either positional space partition only or the positional and normal space partition, outperformed the simple algorithm. The method of partitioning both positional and surface normal space consumed the least amount of time in search of the locally visible points. Table 1 also illustrated the time consumed in search of occlusion-free points for seven viewpoints. The simple algorithm consumed much more time than the k-d tree based method, where a set of potentially occluding points were first sought within the convex hull composed to the boundary box of locally visible points and the sensor point. In the table, there are four views, at which no points lie within the

measurement volume. Therefore, the time for occlusion-free point search is zero.

**Table 1Time comparison for different methods**

**(Time in second)**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Within measurement volume & locally visible | Simple Algorithm | 0.409 | 0.332 | 0.3 | 0.23 | 0.41 | 0.253 | 0.407 |
| | K-d tree with position space partition | 0.27 | 0.228 | 0.089 | 0.053 | 0.27 | 0.015 | 0.379 |
| | K-d tree with positional and normal space partition | 0.12 | 0.086 | 0.045 | 0.016 | 0.175 | 0.014 | 0.175 |
| Occlusion free | Simple Algorithm | 212.14 | 24.27 | 0 | 0 | 24.485 | 0 | 0 |
| | K-d tree based | 0.232 | 0.215 | 0 | 0 | 0.241 | 0 | 0 |

# 5 Conclusion

This paper presents a fast approach for occlusion detection in optical inspection. We first introduced a novel $\delta$ – *occlusion* concept for occlusion check for tessellated part model. We then transformed the point inspectability problem into a range search problem in a k-d tree. We developed several propositions based on the hierarchical positional and normal space partition. The results demonstrated that the algorithm using the partition of positional space of the target objects outperformed the simple algorithm that does not involve space partition. The method using both positional space and normal space partitioning outperformed the method using the positional space partition only.

The contributions of this paper include a novel $\delta$ – *occlusion* concept for occlusion check for point cloud, and the method of hierarchically partitioning both positional space and surface normal space for occlusion detection.

The algorithms can be enhanced in a few ways in the future. We will investigate the sequence of split axis involving both position and surface normal. Currently, a fixed sequence is used and this may not be the best split sequence. We will investigate the optimal size of the points in the leaf node of the k-d tree. We will investigate the use of different boundary box representation, such as discrete orientation polytope, instead of axial-parallel boundary box.

## REFERENCES


Chou, C. Y., Chen, L. L., and Woo, T. C., ‘‘Separating and intersecting spherical polygons: computing machinability on three, four and five-axis numerically controlled machines,’’ *ACM Trans. Graphics* 12 (4), 305–326, 1993.

Elber, G., and Zussman, E., “Cone visibility decomposition of freeform surfaces”, *Computer-Aided Design* 30 (1998) 315–320.

Friedman, J. H., Bentley, J. L., and Finkel, R. A., “An algorithm for finding best matches in logarithmic expected time”, ACM Transactions on Mathematical Software, Vol. 3, No. 3, Sep 1977, pp. 209-226.

Gonzalez-Banos, H.H. and Latombe, J.C., “A Randomized Art-Gallery Algorithm for Sensor Placement”. *Proc. 17th ACM Symp. on Computational Geometry (SoCG'01)*, pp. 232-240, 2001.

Gottschalk S, Lin M, Manocha D, “OBB tree: a hierarchy structure for rapid interference detection”, SIGRAPPH’96 Visual Proceedings, New Orleans, LA, USA 4-9 Aug 1996.

Lin, M.C, and Gottschalk, S., “Collision detection between geometric models: a survey”, Proceedings of IMA Conference on Mathematics of Surfaces, 1998.

Maver, J. and Bajcsy, R., “Occlusions as a Guide for Planning the Next View”, IEEE *Transactions on Pattern Analysis and Machine Intelligence*, Vol. 15, No. 5, May 1993, pp. 417-433.

Nene, S. A., and Nayar, S. K., “A Simple Algorithm for Nearest Neighbor Search in High Dimensions”, Technical Report, Department of Computer Science, Columbia University, 1995.

Petitjean, S., Ponce J., and Kriegman, D. J., “Computing exact aspect graphs of curved objects: Algebraic surfaces”, *International Journal of Computer Vision*, 1992, pp. 9.

Preparata, F. P. and Shamos, M. I., *Computational Geometry: An Introduction*, Springer-Verlag, 1988.

Qian, X. and Harding, K. G., “A Computational Approach for Optimal Sensor Setup”, SPIE Journal *Optical Engineering*, May 2003.

Son, S., Park, H. and Lee, K. H., “Automated laser scanning system for reverse engineering and inspection”, *International Journal of Machine Tools & Manufacture*, Vol. 42, 2002, pp. 889-897.

Tarabanis, K., Allen, P., and Tsai, R., “A survey of sensor planning in computer vision,” *IEEE Trans. Robot. Automat*, vol. 11, pp. 86–104, 1995.

Tarabanis, K., Tsai, R. Y., and Kaul, A., “Computing Occlusion-Free Viewpoints”, IEEE *Transactions on Pattern Analysis and Machine Intelligence*, Vol. 18, No. 3, March 1996, pp. 279 – 292.

Tarbox, G. H. and Gottschlich, S. N., “Planning for Complete Sensor Coverage in Inspection”, *Computer Vision and Image Understanding*, Vol. 61, No. 1, Jan, pp. 84 – 111, 1995.

Trucco, E., Umasuthan, M., Wallace, A. M., and Roberto, V., “Modeling-based Planning of Optimal Placements for Inspection”, IEEE *Transactions on Robotics and Automation*, Vol. 13, No. 2, 1997, pp. 182-194.

Zachmann, G., “Real-time and exact collision detection for interactive virtual prototyping”, 1997 ASME Design Engineering Technical Conferences, Sacramento, CA.