Approximate Distance Queries for Path-planning in Massive Point Clouds

David Eriksson and Evan Shellshear
Fraunhofer-Chalmers Centre, Gothenburg, 412 88, Sweden

Keywords: Point Clouds, Distance Computation, Path-Planning, Simplification.

Abstract: In this paper, algorithms have been developed that are capable of efficiently pre-processing massive point clouds for the rapid computation of the shortest distance between a point cloud and other objects (e.g. triangulated, point-based, etc.). This is achieved by exploiting fast distance computations between specially structured subsets of a simplified point cloud and the other object. This approach works for massive point clouds even with a small amount of RAM and was able to speed up the computations, on average, by almost two orders of magnitude. Given only 8 GB of RAM, this resulted in shortest distance computations of 30 frames per second for a point cloud originally having 1 billion points. The findings and implementations will have a direct impact for the many companies that want to perform path-planning applications through massive point clouds since the algorithms are able to produce real-time distance computations on a standard PC.

1 INTRODUCTION

High-resolution point clouds have become very important in the last decades as researchers have started to exploit their advantages over triangle-based models in computer graphics applications (Tafuri et al., 2012), (Sankaranarayanan et al., 2007). Improvements in scanning technologies make it possible to easily scan very large objects, thereby making point clouds more popular than CAD models in certain applications. This is because point clouds offer the user the ability to acquire valid representations of the real state of the environment under consideration and not just the nominal or planned appearance (Berlin, 2002). Scanned point cloud models are also able to provide up-to-date information about local changes, which is often much easier to acquire than updating a given CAD model. Most importantly, however, for industrial settings that lack a CAD (or similar) model, one can scan an entire factory much easier than building a new CAD model from scratch.

One area where point clouds are useful is in path-planning, where the core algorithms are based on collision detection and/or the computation of the shortest distance between a point cloud and an object moving through the point cloud. The processing and efficient structuring of massive point clouds is critical in order to speed-up path-planning algorithms, (Pauly et al., 2002). The reason for this is that computing the necessary collisions and/or distances is a common bottleneck for path-planning algorithms (Bialkowski et al., 2013).

In certain scenarios, the point clouds used for path-planning can contain billions of points, (Fröhlich and Mettenleiter, 2004), making it a challenge to process them efficiently. Volvo Cars have scanned their factories in Torslanda with the resulting point cloud containing 10 billion points, (Volvo, 2013).

Path-planning through environments consisting of triangle meshes has been studied intensively and there exists a significant amount research on the subject (LaValle, 2006), (Latombe, 1990), (Carlson et al., 2013), (Spensieri et al., 2013), (Spensieri et al., 2008) and (Hermansson et al., 2012). Although the area of path-planning with point clouds is newer, there are methods designed specifically for it, such as (Landa, 2008) and (Landa et al., 2007). Path-planning through hybrid environments has also been considered, where the geometry is a point cloud and the object to be path-planned is a triangle mesh; such combinations have been studied in (Tafuri et al., 2012), (Sucan et al., 2010) and (Pan et al., 2011). Hybrid path-planning is uncommon because of the possibility to triangulate the point cloud and use path-planning on the resulting triangle meshes, (Dupuis et al., 2008). However, this triangulation process is only of interest for point clouds smaller than those considered in this paper, for which the usual algorithms used to triangulate point

Eriksson D. and Shellshear E..
Approximate Distance Queries for Path-planning in Massive Point Clouds.
DOI: 10.5220/0005002000200028
In *Proceedings of the 11th International Conference on Informatics in Control, Automation and Robotics* (ICINCO-2014), pages 20-28
ISBN: 978-989-758-040-6

clouds would be far too cumbersome to be practical, (Tafuri et al., 2012).

Path-planning can be approached from two different directions either via using collision queries or distance queries. In this article we focus on path-planning algorithms that require a fast distance computation. Our results demonstrate the possibility of speeding up distance queries in order to make path-planning for massive point clouds feasible. By using distance queries it is possible to provide more robust results when path-planning in point clouds such as in the aforementioned virtual Volvo Cars factory. In addition, in real life, designers have tolerances and error margins built into their designs that need to be taken into account when path-planning. Such concerns can easily be managed by basing path-planning on computing distances. On the other hand, by using distance based path-planners we can also exploit these error margins to speed up distance computations. If the designer has an error of a couple of centimeters in their models, then it is possible to speed up distance queries because we know that millimeter accuracy is not necessary. With this insight and new ways of using data structures for distance queries, we are able to speed up distance computations for massive point clouds by almost two orders of magnitude. We also present two theorems guaranteeing that our algorithm performs as desired. To the best of the author's knowledge, these two theorems do not exist in the literature.

This paper is structured as follows: Section 2 will describe how a simplified point cloud can be created given a maximal allowed error in the computed shortest distance. We also prove that the proposed simplification methods produce an error in the distance queries that can be controlled by the level of simplification. Section 3 explains how the simplified point cloud can be divided into subsets to expedite distance computations. Section 4 explains how to compute the shortest distance between the simplified point cloud subsets and the path-planning object based on these subsets. Section 5 evaluates the algorithms by considering a real-world point cloud and triangulated geometry. In the final section we conclude. The results in this paper are based on the work in (Eriksson, 2014).

2 SIMPLIFICATION OF THE POINT CLOUD

In order to be able to work more efficiently with a massive point cloud, it would be beneficial to create a simplification of the point cloud with fewer points. The fewer points such a representation has, the faster the distance computations will be. Hence, we look for an appropriate trade-off between a slightly incorrect but much faster distance computation.

2.1 Notation

To prepare for later results, some notation is necessary. The point cloud will always be denoted by P and its corresponding cardinality (number of points) by $|P|$, which will be assumed to be finite. The point cloud consists of a set of points in $\mathbb{R}^n$ and this paper will focus on $n = 2$ or $n = 3$ unless stated otherwise. Denote by p_i, $i = 1, 2, \ldots, n$, the coordinates of a point $p \in \mathbb{R}^n$ and let $p^1, p^2, \ldots, p^m$ denote m points where $p^i \in \mathbb{R}^n$. When a subset of P is considered it will be denoted by Q, i.e. $Q \subseteq P$. Multiple disjoint subsets of P will be denoted by Q_i and when no particular subset is of interest the index will be dropped. The object that will be moved through the point cloud will be denoted by S, and it will be a compact subset of $\mathbb{R}^n$.

We now give a central definition that plays an important role in the rest of the paper, that of the distance between objects.

Definition 1. *Let $Q, R \subset \mathbb{R}^n$ and for $q \in \mathbb{R}^n$ let $\|q\|$ stand for the Euclidean norm. Define:*

$$
\begin{aligned}
d(q,r) &= \|q-r\|,\\
d(q,R) &= \inf_{r\in R} d(q,r),\\
d(Q,R) &= \inf_{\substack{q\in Q\\ r\in R}} d(q,r),\\
d_H(Q,R) &= \max\left\{ \sup_{q\in Q}\inf_{r\in R} d(q,r), \sup_{r\in R}\inf_{q\in Q} d(q,r) \right\},
\end{aligned}
$$

whereby the final distance definition is the well-known Hausdorff metric. The next definition defines what is meant by a simplification of a point cloud.

Definition 2. *For $\varepsilon \in \mathbb{R}_{>0}$ a point cloud P_ε is referred to as an ε-simplification of P if it satisfies:*

1. $|P_\varepsilon| \leq |P|$
2. $d_H(P, P_\varepsilon) \leq \varepsilon$.

The first condition means that an ε-simplification can never have more points than the original point cloud. The second condition means that for all points in the original point cloud P, there should be a point in the ε-simplification near it and also that we do not create new points in the ε-simplification of P that are far away from points in the original cloud P. Both these conditions seem very natural. In Section 2.4 we will show that the second requirement implies the desirable condition that

$$|d(P,R) - d(P_\varepsilon,R)| \leq \varepsilon, \forall R \subseteq \mathbb{R}^n. \qquad (1)$$

We now present two simplification methods that fulfill Definition 2. Note also that one could attempt a simpler down-sampling, however, for our applications we need to guarantee that the down-sampling satisfies Equation 1 and it is not so obvious how to produce an efficient down-sampling that fulfills this requirement.

2.2 The Grid-based Partitioning

The first ε-simplification scheme presented here is based on partitioning the point cloud by small axis-aligned grid boxes with diagonal 2ε, with $\varepsilon \in \mathbb{R}_{>0}$. When simplifying the point cloud, two points that are closer than 2ε may be represented by the midpoint of the line connecting them in P_ε without violating Equation 2 in Definition 2. The grid-based partitioning utilizes this fact by partitioning the original bounding box of the point cloud into disjoint axis-aligned grid boxes with side length $\frac{2}{\sqrt{n}}\varepsilon$ so that the diagonal of each grid box is 2ε. All points that fall into the same axis-aligned grid box may be represented by the box's midpoint in P_ε. In order to allow for this partitioning, the original bounding box of P can easily be extended to make all side lengths divisible by $\frac{2}{\sqrt{n}}\varepsilon$. Extending the bounding box will still guarantee that it contains P.

The simplification scheme is illustrated in Figure 1 where the original point cloud in 2D can be seen to the left and the resulting simplified point cloud obtained from using the grid-based partitioning ε-simplification can be seen to the right.

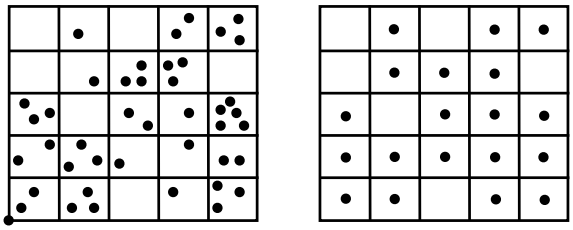

Figure 1: (Left) Original 2D point cloud. (Right) ε-simplification of the point cloud after using the grid-based partitioning.

Note that it is possible that $|P_\varepsilon| = |P|$ if only one point falls within each axis-aligned grid box, which could happen if the point cloud is sparse or if ε is very small. In addition, it is obvious that such a simplification method fulfills our definition of an ε-simplification.

2.3 The k-means Method

The grid-based partitioning described in the previous section has the advantage of having linear time-complexity in the size of the point cloud, but it may give a non-optimal partitioning for at least two reasons:

1. A box of diagonal 2ε can be contained in a sphere with radius ε.
2. The position of the axis-aligned grid boxes is chosen based only on the bounding box of P and no other properties of the point cloud

The first point addresses the fact that an axis-aligned grid box of diagonal 2ε is not the object of maximal volume that can be used to simplify a set of points. In fact, Equation 2 of Definition 2 will also hold in the case when all points falling within a sphere with radius ε are represented by the center of the sphere. The location of the spheres will depend on the distribution of the point cloud since the original bounding box cannot be partitioned into disjoint spheres.

This also addresses the second problem of the grid-based partitioning, which concerns the fact that the axis-aligned grid boxes placement only depends on the bounding box of P and not on the structure of the point cloud. Taking the distribution of points into account is expected to make it possible to create an ε-simplification with a smaller $|P_\varepsilon|$.

One way of placing the spheres adaptively is to use k-means clustering. The k-means method is one of the oldest clustering methods whose aim is to divide a set of points into k clusters. The algorithm is based on work by Lloyd, (Lloyd, 1982), and is often referred to as Lloyd's algorithm. The clusters $C_1, \ldots, C_k$ have the properties $C_i \subseteq Q$, $C_i \cap C_j = \emptyset$ for $i \neq j$, and $\bigcup_{i=1}^{k} C_i = Q$. Each cluster has a centroid μ^i which is computed as the the mean value of all points belonging to cluster C_i. A point is assigned to the cluster with the centroid closest to the point. The k-means method for a set of points Q can be written as the optimization problem:

$$\underset{C_1,\ldots,C_k}{\text{minimize}} \sum_{j=1}^{k} \sum_{i \in C_j} \|q^i - \mu^j\|^2, \forall q^i \in Q. \qquad (2)$$

The k-means method can be used to divide the point cloud recursively. For each new subset it has to be investigated whether the radius of the smallest enclosing sphere has a radius that is no more than ε. If the radius is larger than ε, the k-means method can be used to create new clusters until each cluster can be represented by the center of a sphere with radius of at most ε. An optimal k-means method is expected to produce a better clustering than the grid-based partitioning as is illustrated in Figure 2.

As with the grid-based simplification it is possible that $|P_\varepsilon| = |P|$ if ε is small in comparison to the distribution of points. Otherwise it is obvious that such a

simplification method also fulfills our definition of an ε-simplification.

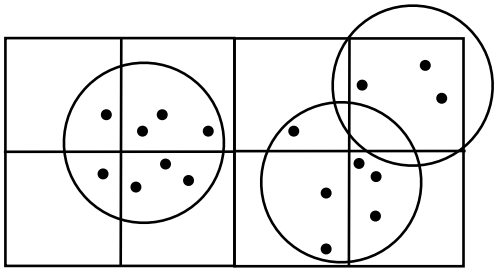

Figure 2: Grid based method compared to the k-means method in 2D, illustrating that the k-means method may be able to create a simplified point cloud with fewer points.

2.4 Maximum Error In the Computed Shortest Distance

This section will show that Condition 2 of Definition 2 implies Equation 1. This is trivial if P consists of only one point and follows directly from the triangle inequality, see Figure 3. The following theorem shows that the error in the shortest distance is always bounded by ε for an ε-simplification scheme satisfying Definition 2.

Theorem 1. *Let P be a point cloud and let P_ε be an ε-simplification of P and let $R \subseteq \mathbb{R}^n$. Then it follows that*

$$|d(P,R) - d(P_\varepsilon,R)| \leq \varepsilon \tag{3}$$

Proof. Let $p = \arg\min_{y\in P} d(y,R)$ and let $p_\varepsilon \in P_\varepsilon$ be such that $d(p,p_\varepsilon) \leq \varepsilon$. It follows that

$$d(p_\varepsilon,R) \leq d(p,p_\varepsilon) + d(p,R) \leq \varepsilon + d(p,R). \tag{4}$$

Let $q_\varepsilon = \arg\min_{y\in P_\varepsilon} d(y,R)$. Then,

$$d(q_\varepsilon,R) \leq d(p_\varepsilon,R) \leq \varepsilon + d(p,R). \tag{5}$$

In addition, there exists a point $q \in P$ such that $d(q,q_\varepsilon) \leq \varepsilon$. Hence,

$$d(q,R) \leq d(q,q_\varepsilon) + d(q_\varepsilon,R) \leq \varepsilon + d(q_\varepsilon,R) \tag{6}$$

and as before we have

$$d(p,R) \leq d(q,R) \leq \varepsilon + d(q_\varepsilon,R). \tag{7}$$

Hence from Equations 5 and 7 it follows that,

$$|d(P,R) - d(P_\varepsilon,R)| \leq \varepsilon. \tag{8}$$

□

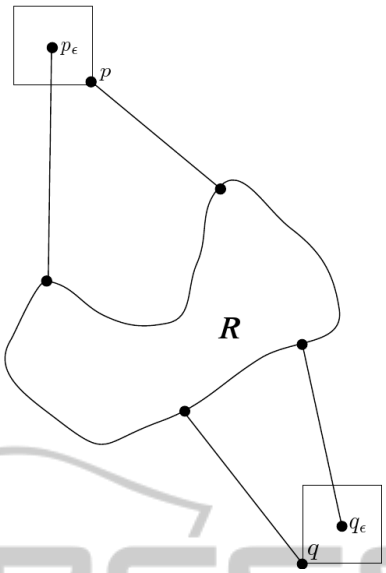

Figure 3: Illustration of the maximal error in Theorem 1 in the case of simplifying with grid boxes with diagonal 2ε.

3 ORGANIZING THE POINT CLOUD FOR FAST DISTANCE QUERIES

This section explains how the original bounding box of the point cloud will be successively divided and organized into new disjoint sub-boxes in order to obtain a partitioning of the original point cloud for fast out-of-core distance queries. These subsets of points make it possible to carry out distance queries for large point clouds given only a small amount of main memory.

Given the original point cloud, it is straightforward to find the bounding box of P. Suppose that we want to divide the point cloud into subsets of at most T points and that $|P| > T$. By splitting the original bounding box along one of the coordinate directions two new sub-boxes can be created. The sub-boxes are divided into two new sub-boxes recursively until all subsets contain no more than T points. Note that there are several options for dividing the sub-boxes, such as along the longest side or the coordinate direction with the largest variance. Each method of subdivision affects the quality of the data structures that are based on these sub-boxes and we chose to split in the coordinate direction with the largest variance. Splitting in this direction seemed to produce good results although more thorough tests are required to investigate the relationship between splitting choice and performance results.

We choose the value of T so that numerous sub-

sets of points can fit in main memory but also so that each subset has a large number of points so we minimize the number of reads from hard disk. At the same time, by choosing T so that we have numerous subsets, we can then exploit this by carrying out the distance queries to each of these subsets in parallel. We explore the effects of different values of T in Section 5. The choice of T obviously depends significantly on the disk transfer rate (SSD vs HDD) and our choices are optimized for our chosen hardware.

4 DISTANCE COMPUTATION

This section describes how the distance from a subset to the object can be computed quickly in-core. It introduces the Proximity Query Package (PQP) and also derives a necessary condition in order to determine when a subset can contain the point closest to the object in the point cloud considered.

4.1 Proximity Query Package

A fast way to find the shortest distance between a point cloud and a triangulated or point-based object is needed in order to be able to compute distances to the subsets that may contain the closest point. In order to do so we chose to use PQP which was designed for collision detection, distance computation, and tolerance verification between pairs of geometric models, (Larsen et al., 1999). It has been shown to be the best choice of bounding volume hierarchy for distance queries when fast queries are the goal, (Larsen et al., 1999), (Lauterbach et al., 2010). PQP uses swept sphere volumes to create a bounding volume hierarchy that can be used to efficiently compute the shortest distance by traversing the tree of bounding volumes.

The memory requirement to build a single PQP model is significant since building a PQP model with 1 billion points uses about 36GB of RAM. This memory limitation is not a problem for the proposed approach due to our memory-based management of the PQP models, which will store numerous small PQP models out of core. Hence each subset of points will have one PQP model, so if T is chosen small enough each individual PQP model can be built without using much memory and can easily be stored out-of-core and read into main memory quickly as disk transfer time is often the bottleneck (see (Eriksson, 2014) for more results).

4.2 STXXL

The Standard Template Library for Extra Large Data Sets (STXXL) is an implementation of the C++ standard template library for external memory (out-of-core) computations. The point cloud will be stored in an STXXL vector (Dementiev et al., 2008) during the simplification phase. However, using STXXL imposes some limitations on users such as only being able to hold so-called plain old data structures (ints, floats, doubles, etc.) as elements in a STXXL vector, (Dementiev et al., 2008). It is also not allowed to use references or pointers to elements in an STXXL vector since elements are temporarily read into memory when requested, so that references will get invalidated when elements are evicted from memory. These limitations makes the construction of a single out-of-core PQP model with STXXL problematic. In addition, the implementation of an out-of-core PQP data structure is not desirable because the changes would require abandoning the advantageous C++ programming paradigm to implement all data structures as plain old data structures. Another reason is that RAM accesses are about one order of magnitude faster than memory accesses using STXXL when data has to be read from the disk.

In order to avoid considering all subsets for each distance query, Theorem 2 in the next section will provide a necessary condition for a subset to contain the point closest to the object. The PQP models that are closest will be kept in-core and PQP models that are far away will remain on the hard disk, in cases where all PQP models may not fit in-core. This situation can change when the object moves, in which case some PQP models will have to be loaded while others will have to be removed from main memory in case all of the PQP models cannot fit at once.

To achieve this, PQP was modified to allow writing of the interior data structure in a binary format to the hard disk. Note that it is difficult to apply such a method to a single large PQP model and use it to only read in parts of the hierarchy on an as needed basis. This is because when querying a bounding volume hierarchy it is unknown which parts of the hierarchy will be required and accesses can be largely random. In the current case, with smaller individual models and the coming Theorem 2, we are able to avoid this problem. Because of this, it was decided to write each PQP model to the hard drive by using a binary format and then reload them when needed.

An advantage with this modification is that the distance computation can be carried out in parallel, which would not have been the case if an STXXL vector of it had been used to hold the PQP models. An-

other advantage with not using STXXL is that when a PQP model has to be read, PQP models that are far away from S can be removed since they are not expected to be used again. In the case of using STXXL it is not clear what will be removed in order to free up the space for the reading.

4.3 A Fast Exclusion Theorem

In this section a very simple criterion is constructed to make it possible to exclude a majority of the point subsets without computing any distances when it is clear that they are too far away from the object S to contain the closest point. First we describe what is meant by a motion of the object that we wish to move through the point cloud. The object S will be moved via rigid body motions $M(t)$, i.e. translations and rotations. A configuration of S is denoted by $S(t) := M(t)S$, which is the position of S at time t. Assume that the shortest distance is computed at times t_i where $t_0 = 0$ and $S = S(t_0)$. If $p \in S$ is a point that originally belongs to S, then we denote this point at time t after rotations and translations by $p(t) := M(t)p$. Note that p never moves with respect to S. Some definitions are necessary before stating the fast exclusion theorem:

Definition 3. *Define by α_i an upper bound on the largest displacement of S between t_{i-1} and t_i, that is*

$$\alpha_i = \max_{p \in S} \|p(t_i) - p(t_{i-1}))\|. \tag{9}$$

Definition 4. *Define $d_{\min}(t)$ to be the shortest distance from the object $S(t)$ to the point cloud P at time t.*

The idea is that α_i can be used to bound how much closer a subset can be to the object compared to the previous iteration. The result is stated in the following Theorem:

Theorem 2. *Let P be a point cloud and divide P into m disjoint subsets. Let $i \geq 1$, $j \in \{1,\ldots,m\}$ and $d_j(t_{i-1})$ be a lower bound on the distance from subset j to $S(t_{i-1})$. Then no point in subset j can be closer to $S(t_i)$ than*

$$\max(0, d_j(t_{i-1}) - \alpha_i) \tag{10}$$

and also

$$d(P,S) \leq d_{\min}(t_{i-1}) + \alpha_i. \tag{11}$$

Proof. The first statement follows from the fact that the object has moved at most α_i and that no point in point subset j was closer than $d_j(t_{i-1})$ before S moved, which yields that no points in the subset can be closer than $\max(0, d_j(t_{i-1}) - \alpha_i)$. Since the object has moved at most α_i, the point that was previously at a distance $d_{\min}(t_{i-1})$ away cannot be further away than $d_{\min}(t_{i-1}) + \alpha_i$. This implies the second statement. □

The theorem makes it possible to neglect a large amount of the subsets that are known to be too far away to contain the point closest to the object. The idea is to compute the initial values $d_j(0)$ before the object starts moving. Similarly, an upper bound on $d_{\min}(0)$ can be computed by considering only a few points per subset and save the shortest distance found. The use of the theorem is illustrated in Algorithm 1.

Algorithm 1: Computation of the shortest distance using Theorem 2.

```
for subset number j do
    Compute an initial lower bound of d_j(0)
end for
Find an upper bound for d_min(0)
i ← 1
while Object is moving do
    Move S to the next position and compute α_i
    d_min(t_i) ← d_min(t_{i-1}) + α_i
    for each subset number j do
        d_j(t_i) ← d_j(t_{i-1}) − α_i
        if d_j(t_i) ≤ d_min(t_i) then
            Compute d_j(t_i) using the PQP model
            Set d_min(t_i) = min(d_j(t_i), d_min(t_i))
        end if
    end for
    i ← i + 1
end while
```

This inequality based on α_i is designed for quick evaluation, however it should be mentioned that it can be very weak in the sense that it does not take the direction of movement into account. If the object is moved away from a subset so that the distance from the points in subset j to the object increases, the lower bound will be pessimistic since α_i will be subtracted from $d_j(t_{i-1})$.

5 SIMULATION RESULTS

The point cloud that will be considered is a part of a factory and consists of about 1 billion points. In this section of the factory a car chassis is required to move from a start to an end configuration. In addition, it is often required that the car be at least a given distance from the surrounding point cloud along the whole path. A picture of the point cloud can be seen in Figure 4. The object will be moved from the hub in the upper right corner to the hub in the lower left

corner of Figure 4 along a known path. There are a total of 700 discrete time steps. The spacing between points in the point cloud is about 0.5 centimeters.

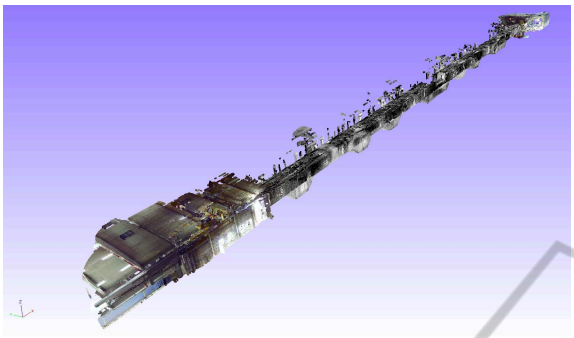

Figure 4: An overview of the original point cloud consisting of about 1 billion points.

The object that is moved through the point cloud is a holding mechanism for a car and its triangulation consists of 78,403 triangles and can be seen in Figure 5. The position of the object along the path is given at discrete points in time and the displacements of the object vary in size. The variation in the displacements will directly affect the computed values of α_i.

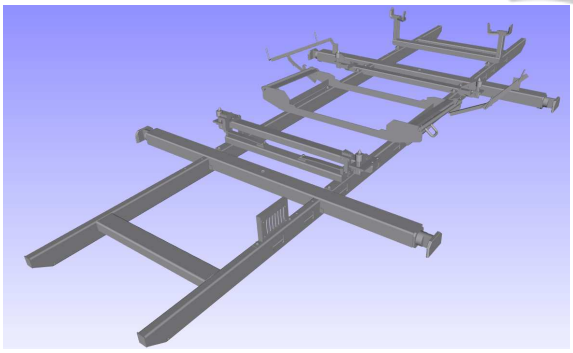

Figure 5: A visualization of the object, a car holding mechanism, that will be moved through the point cloud.

To compute α_i we computed the original bounding box of the object S and transformed its eight corners at each iteration and then computed the furthest a corner point had moved and set that equal to α_i. In the case of no rotations of the object this upper bound is in fact exact.

The computer used for the simulations was an Intel Core i7 processor, 32 GB RAM, and a 250 GB SSD. The amount of RAM was constrained to 8 GB in order to force the algorithms to run out-of-core for the original point cloud with 1 billion points. In the first step we built PQP models for both the triangulated object S and the point cloud considered. If all points are stored as triplets of floats, the total memory requirement for all the point cloud PQP models equals 36 GB, so that only at most 22% of the PQP models can reside in memory at the same time. The shortest distances were computed for different values of T (where T was defined in Section 3) and the fastest time was obtained for $T = 1,000,000$ for which the distance computations took a total of 500s, which corresponds to 1.4s per distance computation (see (Eriksson, 2014) for results on other values of T). Theorem 2 managed to rule out, on average, 96% of the subsets for each distance computation which gave a significant speed-up compared to if all subsets had been considered.

This is too slow for many path-planning applications (especially ones that are required to run in real time) so a simplified point cloud was created instead in order to speed up the distance computations. In order to compare the two ε-simplification methods from Section 2 the original point cloud was simplified for different values of ε and the results can be seen in Table 1.

It is clear that the larger ε is, the smaller the number of points in the simplified point cloud is. It can also be seen that the k-means method outperforms the grid-based partitioning in the sense that it is able to create an ε-simplification with fewer points. On the other hand, it should be mentioned that the k-means method is significantly slower than using the grid-based partitioning, so there is a trade off between obtaining a small value of $|P_\varepsilon|$ and the execution time of the simplification phase.

Table 1: Number of points (millions) and percentage of points remaining for each simplification method.

ε	k-means, $k=2$		k-means, $k=3$		Grid boxes	
(meters)	Points	%	Points	%	Points	%
0.005	813.6	79.4	874.6	85.4	965.4	94.3
0.01	467.2	45.6	556.5	54.3	826.4	80.6
0.02	205.1	20.0	241.0	23.5	378.6	37.0
0.03	111.2	10.8	129.8	12.7	181.2	17.7
0.05	47.5	4.6	55.6	5.4	69.9	6.8
0.1	13.4	1.3	16.2	1.6	18.9	1.8

When simplifying the point cloud we chose $\varepsilon = 2$ centimeters to correspond to a realistic tolerance for the path-planning applications where these techniques will be used. With $\varepsilon = 2$ cm, the k-means method yielded a simplified point cloud with 200 million points, so that the memory required for the PQP models was about 7.2 GB and will therefore fit in-core with the memory restriction of 8 GB. When comparing the time taken for each method, the k-means simplification method is around two orders of magnitude slower than the much simpler grid box method. More results can be found in (Eriksson, 2014). The distances were computed in-core to the simplified point cloud and for $T = 1,000,000$ the total execution time

was 22.1 seconds, which means that each distance computation took on average 0.03s which is a significant speed-up compared to the original point cloud and would allow for real-time computation of distances (i.e. for frame rates of 20 - 30 frames per second). A comparison between the time for each distance computation for the original and the simplified point clouds can be seen in Figure 6. As can be seen in the figure, the new distance computations are orders of magnitude faster in the simplified point cloud than in the original point cloud.

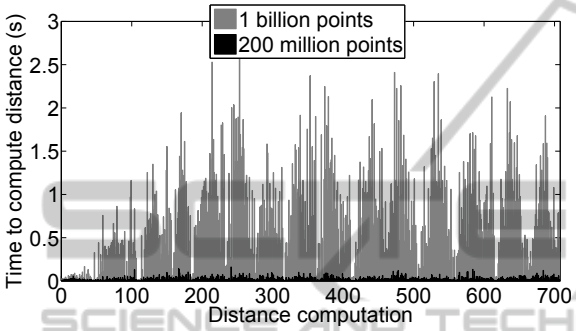

Figure 6: Time for distance computations for the original and the simplified point clouds.

6 CONCLUSIONS

It has been verified that an in-core PQP implementation can be used even for massive point clouds by dividing the point cloud into subsets and then use Theorem 2 in order to rule out many of the subsets. Theorem 2 was able to rule out 96% of the subsets so that only a few subsets had to be considered for each distance computation. Theorem 2 has a drawback of not being precise in some cases and therefore forcing reads of subsets that are far away from the object. This could be avoided by using a fast distance approximation scheme based on the bounding boxes of the subsets or the extreme points of the convex hulls in order to approximate how close subsets actually are before considering all the points. This is appealing since reading the subset from the hard-disk is the major bottleneck in the proposed approach which is why it is of interest to use a few points to approximate the distance to the rest of the points.

By simplifying the point cloud using either the grid-based partitioning or the k-means clustering method, a simplified point cloud was created that was easier to work with. It was seen in Section 5 that a point cloud with 1 billion points could be simplified, with $\varepsilon = 2$ cm, to a point cloud with 200 million points to which the distances could be computed in-core given 8 GB of memory and therefore decrease the computation time from 500s to 22s. By allowing the computed shortest distance to deviate by ε, it is therefore possible to decrease the time for distance computations significantly. It was also seen that the k-means method provides a simplified point cloud with a smaller $|P_\varepsilon|$ than the grid-based partitioning, even though it is significantly slower.

Because methods used to carry out collision detection with massive models are inappropriate for distance queries, (Yoon et al., 2004), this work fills a gap in the literature for distance computations in massive point clouds. Although methods exist for collision detection in point clouds, (Pan et al., 2011), their extension to massive point clouds is not clear. In addition, given the minimum distance restrictions often required during path planning, using collision detection does not seem to be natural. Hence, we have shown that the path-planning problem requiring minimum distances can be solved directly via distance computations.

Although the method has been shown to be fast, distance computations are in general slower than identical collision queries. Hence, if the user's goal is simply the fastest computations possible and he or she does not require the extra distance information, then it might be beneficial to use a collision based approach.

In future work we would like to address the following promising avenues for research:

- Use the extreme points of the convex hull of each subset in order to approximate the distance to the object when Theorem 2 fails to rule out a subset.
- Take the direction of motion of S into account in Theorem 2.
- Use bounding boxes instead of swept sphere volumes in PQP in order to decrease the memory requirement for a PQP model.
- Investigate how to choose T optimally as a function of amount of RAM, $|P|$ and the properties of the disk(s) used.

ACKNOWLEDGEMENTS

This work was carried out at the Wingquist Laboratory VINN Excellence Centre, and is part of the Sustainable Production Initiative and the Production Area of Advance at Chalmers University of Technology. It was supported by the Swedish Governmental Agency for Innovation Systems.

REFERENCES

Berlin, R. (2002). Accurate robot and workcell simulation based on 3d laser scanning. In *Proceedings of the 33rd ISR (International Symposium on Robotics)*.

Bialkowski, J., Karaman, S., Otte, M., and Frazzoli, E. (2013). Efficient collision checking in sampling-based motion planning. In *Algorithmic Foundations of Robotics X*, pages 365–380. Springer.

Carlson, J. S., Spensieri, D., Söderberg, R., Bohlin, R., and Lindkvist, L. (2013). Non-nominal path planning for robust robotic assembly. *Journal of manufacturing systems*, 32(3):429–435.

Dementiev, R., Kettner, L., and Sanders, P. (2008). Stxxl: standard template library for xxl data sets. *Software: Practice and Experience*, 38(6):589–637.

Dupuis, E., Rekleitis, I., Bedwani, J.-L., Lamarche, T., Allard, P., and Zhu, W.-H. (2008). Over-the-horizon autonomous rover navigation: experimental results. In *International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS), Los Angeles, CA*.

Eriksson, D. (2014). Point cloud simplification and processing for path-planning. Master's thesis, Chalmers University of Technology.

Fröhlich, C. and Mettenleiter, M. (2004). Terrestrial laser scanning—new perspectives in 3d surveying. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36(Part 8):W2.

Hermansson, T., Bohlin, R., Carlson, J. S., and Söderberg, R. (2012). Automatic path planning for wiring harness installations (wt). In *4th CIRP Conference on Assembly Technology and Systems-CATS 2012, University of Michigan, Ann Arbor, USA on May 21-23, 2012*.

Landa, Y. (2008). *Visibility of point clouds and exploratory path planning in unknown environments*. ProQuest.

Landa, Y., Galkowski, D., Huang, Y. R., Joshi, A., Lee, C., Leung, K. K., Malla, G., Treanor, J., Voroninski, V., Bertozzi, A. L., et al. (2007). Robotic path planning and visibility with limited sensor data. In *American Control Conference, 2007. ACC'07*, pages 5425–5430. IEEE.

Larsen, E., Gottschalk, S., Lin, M. C., and Manocha, D. (1999). Fast proximity queries with swept sphere volumes. Technical report, Technical Report TR99-018, Department of Computer Science, University of North Carolina.

Latombe, J.-C. (1990). *ROBOT MOTION PLANNING.: Edition en anglais*. Springer.

Lauterbach, C., Mo, Q., and Manocha, D. (2010). gproximity: Hierarchical gpu-based operations for collision and distance queries. In *Computer Graphics Forum*, volume 29, pages 419–428. Wiley Online Library.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge university press.

Lloyd, S. (1982). Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137.

Pan, J., Chitta, S., and Manocha, D. (2011). Probabilistic collision detection between noisy point clouds using robust classification. In *International Symposium on Robotics Research*.

Pauly, M., Gross, M., and Kobbelt, L. P. (2002). Efficient simplification of point-sampled surfaces. In *Visualization, 2002. VIS 2002. IEEE*, pages 163–170. IEEE.

Sankaranarayanan, J., Samet, H., and Varshney, A. (2007). A fast all nearest neighbor algorithm for applications involving large point-clouds. *Computers & Graphics*, 31(2):157–174.

Spensieri, D., Bohlin, R., and Carlson, J. S. (2013). Coordination of robot paths for cycle time minimization. In *CASE*, pages 522–527.

Spensieri, D., Carlson, J. S., Bohlin, R., and Söderberg, R. (2008). Integrating assembly design, sequence optimization, and advanced path planning. *ASME Conference Proceedings*, (43253):73–81.

Sucan, I. A., Kalakrishnan, M., and Chitta, S. (2010). Combining planning techniques for manipulation using realtime perception. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2895–2901. IEEE.

Tafuri, S., Shellshear, E., Bohlin, R., and Carlson, J. S. (2012). Automatic collision free path planning in hybrid triangle and point models: a case study. In *Proceedings of the Winter Simulation Conference*, page 282. Winter Simulation Conference.

Volvo (2013). Volvo Cars in Gothenburg, Personal communication.

Yoon, S.-E., Salomon, B., Lin, M., and Manocha, D. (2004). Fast collision detection between massive models using dynamic simplification. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 136–146. ACM.