Fast Point to Mesh Distance by Domain Voxelization

Geordan Gutow and Howie Choset

Abstract— **Computing the distance from a point to a triangle mesh is a key computational step in robotics pipelines such as registration and collision detection, with applications to path planning, SLAM, and RGB-D vision. Numerous techniques to accelerate this computation have been developed, many of which use a cheap pre-processing step to construct a hierarchical decomposition of the mesh. If the mesh is fixed and known ahead of time, there is an opportunity to conduct more expensive pre-computations to accelerate the subsequent distance queries. This work presents a voxelization approach, implemented on both CPU and GPU, to compute point to mesh distance that constructs for each voxel a near-minimal set of triangles that is guaranteed to include every triangle that is closest to at least one point in the voxel. Theoretical and numerical comparisons with six alternative distance algorithms demonstrate the speed advantages of the proposed method.**

I. Introduction

Computing the distance from a point to the surface of a triangular mesh is a key step in many robotics algorithms, including collision checking [1]–[4] and point cloud registration [5]–[8]. In many applications, distance computations represent a large part of the runtime expense of the algorithm. A variety of approximate point to mesh distance techniques exist that efficiently pre-compute the distance to the mesh at numerous selected points in the domain, allowing very fast approximate distance queries via interpolation. This includes scan-conversion as in [9], Eikonal solvers like [10], and dedicated distance transform algorithms like [11].

For fast exact distance computations, the standard solution is to identify a subset of the triangles in the mesh that may be closest to the query point and compute the distance to all of them. Data structures such as k-d trees [12], [13], octrees [7], binary space partitions [2], or bounding volume hierarchies [1], [4], [12] are commonly used to accelerate these queries by pruning triangles that cannot contain the closest point. The "ideal" data structure for such acceleration is a representation of the Generalized Voronoi Diagram (GVD) of the mesh, which labels locations with their closest triangle. With the GVD, a distance query reduces to looking up the nearest triangle and computing a single point to triangle distance. Constructing and storing the exact, or even approximate, GVD of a triangle mesh is challenging [14]–[16].

Instead, one may discretize the domain into voxels and precompute for each voxel a set of triangles guaranteed to include all triangles that are closest to *any* one point in the voxel. Similar tactics are used in [2], [8], [17]. A distance query then requires testing the point against every triangle in the set. The query speed of a voxelization scheme is therefore dominated by the size of the set. In general, one can reduce the size of this set by shrinking the size of the voxels at the cost of increased memory and construction time. Our contribution is an algorithm to compute for a voxel a near-minimal set of triangles that is nevertheless guaranteed to include the triangle closest to any given point in the voxel. An example set is shown in figure 1. Proofs are presented that the resulting set is no larger than that obtained by the approaches of [8] and [2]. Numerical results are provided showing significantly faster distance queries than the voxelizations of [2], [17], as well as than standard space partitioning and bounding volume hierarchy techniques. A GPU implementation of the proposed approach is demonstrated, running orders of magnitude faster than the CPU version. CPU and GPU implementations of the voxelized distance query are available on Github: https://github.com/biorobotics/point2mesh-prune.

All authors are with the CMU Robotics Institute. Emails: {ggutow, choset}@andrew.cmu.edu. This work was supported by AFRL and the AFOSR.

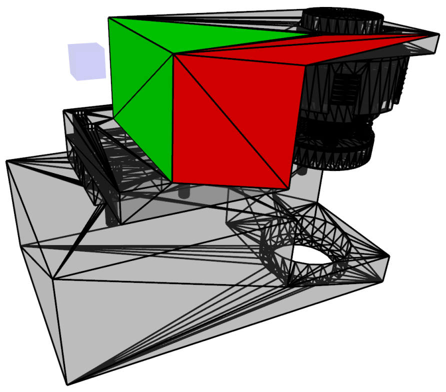

Fig. 1. The green triangles are those that must be tested for points in the blue voxel. In red are the triangles the proposed algorithm proved do not need to be included.

II. Related Work

Hierarchical structures are commonly used to identify a subset of the triangles of the mesh that may be closest to the query point. This has significant overlap with collision testing and ray-tracing, and often a solution to one problem can also be used to solve the other two. An idea common to many works is that of the bounding volume hierarchy (BVH): a simple shape enclosing the full geometry is recursively subdivided into smaller enclosures. Larssen [1] proposed using a tree of Rectangle Swept Spheres (RSS) for the distance query problem, as such shapes can more tightly fit geometries such as triangles than alternatives like axis-aligned bounding boxes (AABB). Lauterbach et al. present efficient GPU implementations of tight fitting BVHs on the GPU, including both RSS and oriented bounding box volumes [3]. Much more recently, Li et al. describe RSS fitting and splitting methods that better handle triangles of widely varying face sizes [4]. This algorithm is incorporated into the commercial IPS path planner.

The tight fit of RSSs comes at the cost of complicated tree construction, so other bounding volumes remain in use. The popular Computational Geometry Algorithms Library (CGAL) incorporates an AABB tree for both distance and intersection queries, which leverages a kd-tree of points on the surface of the mesh to further accelerate distance queries by providing a cheap upper bound on the distance [12]. AABB tree distance queries are used by the R-LOAM and MA-LOAM SLAM pipelines for comparing observed point clouds with the reference model [18], [19].

The kd-tree is a special case of a binary space partition (BSP) tree [20]. Where a BVH splits primitives by creating volumes that enclose ever smaller subsets of the primitives, a spatial partition instead selects hyperplanes that divide the domain and records which side the primitives lie on. In a k-d tree these planes are restricted to be perpendicular to the axes. BSP trees can also be used directly for distance queries as in [2], in which the planes are selected from the faces of the mesh. Spatial partitions need not be binary; the octree and its higher dimensional generalizations are spatial partitions in which the separating hyperplanes are axis aligned but $2^{\text{dimension}}$ partitions are made at once rather than just 2. Octrees are used for distance computations in the Hausdorff distance calculation algorithm of [21] and the point cloud to mesh registration pipeline described by [7]. Drost shows how using a hash table to index the octree reduces the overhead of traversing the tree for point cloud to point cloud nearest neighbor queries in [22].

Voxel decompositions have previously appeared for several applications. The VCG Library [23] implements a number of spatial index data structures that can be used for point to mesh distance queries, including a voxel grid that was used to accelerate the one-sided Hausdorff distance calculation in the Metro tool for comparing two meshes [17]. The Hausdorff distance calculation in [21] leverages a voxel decomposition as well as the octree mentioned earlier. The one-sided triangle to quad mesh Hausdorff distance calculation of [24] similarly fuses both a bounding volume hierarchy and a voxel decomposition to efficiently find the nearest quad element to a query point. Hauth et al. describe a point to mesh distance and collision query algorithm that voxelizes the domain and records links from empty voxels to the non-empty voxels containing the relevant triangles [2]. Recently, Mejia-Parra et al. [8] applied a perfect spatial hash [25] to reduce the memory footprint of a voxel grid for pointcloud to mesh registration via ICP; this maintains constant time look-up of nearby triangles while eliminating the memory cost of empty cells. The tradeoff is that points not within voxels that intersect the mesh surface cannot have their distances computed.

III. Preliminaries

Three vertices $v_i \in \mathbb{R}^3$ define a triangle $T \subset \mathbb{R}^3$. Consider a rigid triangular mesh M defined as a set of triangles. The surface of the mesh $\mathbb{S} = \bigcup_{T_i \in M} T_i \subset \mathbb{R}^3$. The task is to compute the shortest distance d^* between a query point $q \in D \subset \mathbb{R}^3$, and any point $c \in \mathbb{S}$:

$$d^* = \min_{T \in M} \min_{x \in T} ||x - q||_2 \tag{1}$$

M need not be watertight. For triangle T, $d(T, q) = \min_{x \in T} ||x - q||_2$ can be obtained by computing the closest point x (using e.g. algorithmClosestPtPointTriangle from [26]), or slightly more efficiently via direct calculation. Suppose q is restricted to lie in some axis-aligned cube with side length Δ (i.e. a voxel $V \subset D$). We seek a small set of triangles $C \subseteq M$ such that $\min_{T \in C} d(T, q) = d^*$. Let $S_\phi(\rho)$ be the sphere of radius ρ centered on the voxel ϕ, and let $B_\phi(\rho)$ be the ball of radius ρ centered on the voxel ϕ.

IV. Method

Algorithm 1 GetTriangles

```
Require: the corner $c_m \in \mathbb{R}^3$ of $V$ with smallest coordinates, side length $\Delta$, mesh $M$
Ensure: $T \in C_2^*$ if $\exists p \in V : T = \text{argmin}_{\tau \in M} d(\tau, p)$
  corners$\leftarrow$ the corners of $V$
  $I \leftarrow \{T_i \in M : T_i \bigcap V \neq \varnothing\}$
  if $I$ is $\varnothing$ then
      $I \leftarrow \text{argmin}_{T \in M} \min_{y \in V} d(T, y)$
  end if
  $l_1 \leftarrow \min_{T \in I} \max_{y \in \text{corners}} d(T, y)$, $s_1 \leftarrow \Delta + 2l_1$
  $b_1 \leftarrow$ axis aligned box centered at $c_m + <\Delta, \Delta, \Delta>/2$ with side length $s_1$
  $C_1 \leftarrow \{T_i \in M : T_i \bigcap B_1 \neq \varnothing\}$
  $l_2 \leftarrow \min_{T \in C_1} \max_{y \in \text{corners}} d(T, y)$
  $C_2^* \leftarrow \{T \in C_1 : l_2 \geq \min_{y \in V} d(T, y)\}$
```

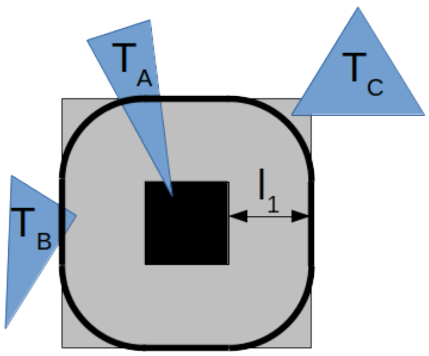

Fig. 2. A 2D projection of the pruning process. Triangle T_C is provably not closest to any point in the black voxel and so can be discarded.

The proposed method, presented in pseudocode as Algorithm 1 and schematically in figure 2, obtains a tight upper bound on the shortest distance from any point in V to the mesh, then discards all triangles that are no closer to any point in V than that bound.

First collect all triangles that intersect the current voxel: $I = \{T_i \in M : T_i \bigcap V \neq \varnothing\}$. If I is empty, add any one triangle from the mesh. A good choice is the triangle whose minimum distance to the boundary of V is smallest. Then compute the one-sided Hausdorff distance from the triangles in I to the voxel:

$$l_1 = \min_{T \in I} \max_{y \in V} d(T, y) \tag{2}$$

Lemma 1: The maximization in $\min_{T\in I} \max_{y\in V} d(T, y)$ need consider only the corners of V.

Proof:

This follows from the convexity of the cube: Assume $q \in V$ is farther from $x \in T \in M$ than all corners of V. Then, all corners lie on the interior of $S_x(q)$, the sphere centered at x intersecting q. V is convex, so all points in V, including q, must also be on the interior of $S_x(q)$. But by assumption q lies on the surface of $S_x(q)$. The assumption that q is farther from x than all corners is falsified.

So for any point x on a triangle T, the furthest point in V from x is not farther than at least one of the corners of V. Thus $\max_{y\in V} d(T, y) = \max_{y\in \text{corners}(V)} d(T, y)$. ■

So in practice, compute the distance from each triangle in I to each of the corners of V, and set l_1 to the largest distance found. Then every point in V is within l_1 of at least one triangle in I, so l_1 is an upper bound on the shortest distance from any point in V to the mesh. In Figure 2, $I = \{T_A\}$ and l_1 is the distance from T_A to the bottom right corner of the voxel (dark square).

Let C_1^* be the set of triangles that would be retained using the l_1 bound. C_1^* consists of the triangles intersecting the cube swept sphere formed from the union of V and a sphere of radius l_1 whose center is swept over the boundary of V. The outline of C_1^* is shown by the wide black line in figure 2. A simple, conservative approximation C_1 is every triangle intersecting the axis aligned bounding box of the cube swept sphere, which is a cube whose faces are l_1 away from the nearest face of V. This box, b_1, is a cube of side length $\Delta + 2 * l_1$, and so $C_1 = \{T_i \in M : T_i \bigcap b_1 \neq \varnothing\}$. Note that $C_1 \supseteq I$. b_1 is shown in figure 2 as the light square, and C_1 contains T_A, T_B, T_C.

This naturally suggests a new bound l_2:

$$l_2 = \min_{T \in C_1} \max_{y \in V} d(T, y) \tag{3}$$

Again, in practice only the corners of V need be considered. Note that l_2 is the tightest possible bound of this form:

Lemma 2: $l_2 = \min_{T\in M} \max_{y\in V} d(T, y)$

Proof: Let $T_m \in M : T_m \notin C_1$. Assume $\max_{y\in V} d(T_m, y) < l_2$. Then $\min_{y\in V} d(T_m, y) \leq l_2 \leq l_1$. But by construction, C_1 contains all triangles T s.t. $\min_y \in V d(T, y) \leq l_1$, which contradicts the assumption that T_m was not in C_1. So no triangle T_m exists in M that yields $\max_{y\in V} d(T_m, y) < l_2$. ■

This bound yields the final set of triangles, $C_2^* = \{T \in C_1 : l_2 \geq \min_{y\in V} d(T, y)\}$, consisting of those triangles in C_1 whose minimum distance to the voxel is not more than the bound. In figure 2, $l_2 = l_1$ and so T_C will not be included in C_2^*, but T_B will.

V. Comparison of Voxelization Schemes

First, consider the voxelization in [8]. Query points farther from the mesh than Δ are considered "outliers" and the voxelization need not produce correct distances for such points. As a result, only voxels such that $D \cap \mathbb{S} \neq \varnothing$ are needed. Each voxel is assigned the triangles that intersect it or an adjacent voxel. This is identical to the set C_1 for $l_1 = \Delta$. The proposed method, if distances greater than Δ may be ignored, must produce $l_1 = \Delta$ at worst and can have $l_1 < \Delta$: a triangle bisecting the voxel parallel to a face yields $l_1 \leq \Delta/2$. Moreover, $C_2^* \subseteq C_1$. Thus in the worst case the same triangle sets are obtained, and for some voxels a smaller triangle set will be found by the proposed method.

A more general voxelization scheme, dubbed the "Linked Voxel Structure," is presented in [2]. The "extended" version is specific to collision detection. The Linked Voxel Structure distinguishes between empty and non-empty voxels:

Definition 1 (Non-empty Voxel): A voxel V is called "non-empty" if its circumscribed sphere $S_V(\frac{\sqrt{3}}{2}\Delta)$ intersects at least one triangle in M.

Note that this a slightly stronger condition than the non-intersection used in [8]. A non-empty voxel is assigned all triangles C_L intersecting $S_V(3\frac{\sqrt{3}}{2}\Delta)$.

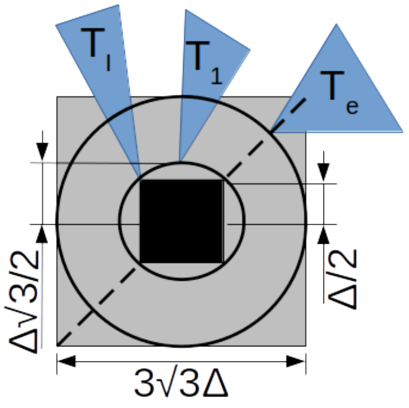

Fig. 3. Schematic showing the triangles used to prove that for a non-empty voxel $C_2^* \subseteq C_L$. The black square is the voxel, the small black circle is the circumscribed sphere of the voxel, and the large black circle is the sphere within which the linked voxel structure stores triangles. The grey square is the largest possible box b_1.

Lemma 3: For a non-empty voxel V, $C_2^*(V) \subseteq C_L(V)$.

Proof: Suppose voxel V is non-empty. Then $\exists T_1 \in M$ in the circumscribed sphere of V (see figure 3). Therefore, $\max_{y\in V} d(T_1, y) \leq \sqrt{3}\Delta$. If $\exists T_I \in M$ intersecting V, $l_1 \leq \sqrt{3}\Delta$ as well. And if no intersecting triangles exist, the proposed method will set l_1 based on the triangle closest to the surface of V. l_1 can be no larger than the distance from the closest triangle to the surface, plus the diagonal of V. The closest triangle is no farther away than T_1 is, and $\min_{y\in V} d(T_1, y) = (\sqrt{3}-1)\frac{\Delta}{2}$. So $l_1 \leq (\sqrt{3}-1)\frac{\Delta}{2} + \sqrt{3}\Delta = (3\sqrt{3}-1)\frac{\Delta}{2}$. b_1 then has edge length $\leq 3\sqrt{3}\Delta$; the sphere used by the Linked Voxel structure is the inscribed sphere of the largest possible b_1, so $C_1 \supseteq C_L$ in the worst case. Suppose $\exists T_e \in C_1 : T_e \notin C_L$. T_e must be in one of the "corner regions" of b_1 that are outside the inscribed sphere; the minimum distance from T_e to the voxel occurs if T_e lies on the diagonal of b_1, almost touching the surface of the inscribed sphere. Thus $\min_{y \in V} d(T_e, y) > 3\frac{\sqrt{3}}{2}\Delta - \frac{\sqrt{3}}{2}\Delta = \sqrt{3}\Delta$. If T_I exists then this is strictly greater than $l_1 \geq l_2$ and so T_e is not in C_2^*. If a triangle does not intersect V, note that T_1 must be in C_1 and so $l_2 \leq \sqrt{3}\Delta$. Again T_e is not in C_2^*. Therefore, $C_2^* \subseteq C_L$. ■

The linked voxel structure handles "empty" voxels by recording a set of nearby non-empty voxels whose associated triangles include every triangle closest to a point in V. This set is determined by first finding the smallest $r \in \mathbb{N}$ such that at least one voxel intersecting $S_V(\Delta r)$ is non-empty. The original paper then recorded all voxels that are non-empty and intersect $B_V(\Delta(r+3))$, claiming that this is sufficient.

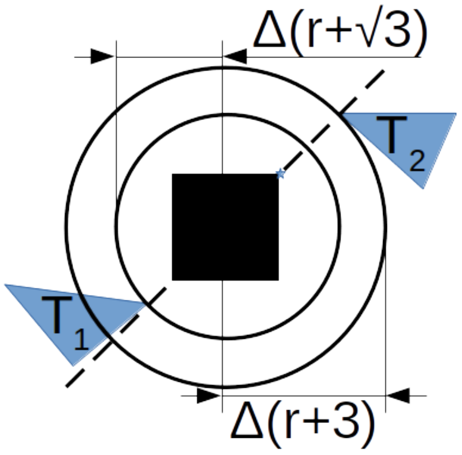

Fig. 4. A scenario in which the linked voxel structure presented in [2] misses a triangle. T_2 is the closest triangle to the starred point, but is not linked to.

This is insufficient for certain cases. As [2] correctly notes, the triangle in the first located non-empty voxel establishes an upper bound of $\Delta(r + \frac{3}{2}\sqrt{3})$ on the distance from a point in V to the mesh. However it is not sufficient to consider triangles up to a distance of $\Delta(r+3) > \Delta(r + \frac{3}{2}\sqrt{3})$ from the *center* of V. Instead one must capture all triangles within $\Delta(r + \frac{3}{2}\sqrt{3})$ of the *surface* of V. Consider a mesh containing two triangles lying on the extended diagonal of V. The scenario is shown in figure 4. The first triangle T_1 is at exactly radius $\Delta(r + \sqrt{3})$, while the second T_2 is on the opposite side of V at radius arbitrarily larger than $\Delta(r+3)$. T_2 will not be captured by the linked voxel structure. The distance from T_1 to the corner closest to T_2 is $\Delta(r + \frac{3}{2}\sqrt{3})$, precisely the upper bound identified by [2]. But the distance from T_2 to that corner can be as little as $\Delta(r+3-\frac{\sqrt{3}}{2})$, which is smaller. For correctness the linked voxel structure should link to the set $L_V = \{N \in D : N \cap B_V(\Delta(r+4)) \neq \varnothing \ \& \ C_L(N) \neq \varnothing\}$, non-empty voxels intersecting $B_V(\Delta(r+4))$.

Lemma 4: For an empty voxel, $C_2^* \subseteq C_L(V) = \bigcup_{N \in L_V} C_L(N)$, the triangles in the corrected linked voxels.

Proof: Let Δr, $r \in \mathbb{N}$, be the radius at which the first non-empty voxel, N_1, was found. Let T_1 be a triangle in the circumscribed sphere of N_1. The distance from a point in V to M is not larger than the distance from that point to T_1, so $l_2 \leq l_1 \leq \max_{y \in V} d(T_1, y) \leq \Delta(r + \frac{3}{2}\sqrt{3})$, where the upper bound is obtained for T_1 along the diagonal of N_1 (of length $\sqrt{3}\Delta$) from the discrete sphere (a further $r\Delta$ to the center of V), and N_1 lies along the diagonal of V_1 (an additional $\frac{\sqrt{3}}{2}\Delta$ to the far corner of V). Therefore if $T_i \in C_2^*$ then $\min_{y \in V} d(T_i, y) \leq \Delta(r + \frac{3}{2}\sqrt{3})$.

Let $L' = \{T \in M : T \cap B_V(\Delta(r+4)) \neq \varnothing\} \subseteq C_L(V)$. $T_i \notin L' \implies \min_{y \in V} d(T_i, y) > \Delta(r+4) - \frac{\sqrt{(3)}}{2}\Delta$, which is $>$ the upper bound for a triangle in C_2^*. So any triangle in C_2^* is also in L' and therefore also in C_L. ■
A schematic for the proof is shown in figure 5.

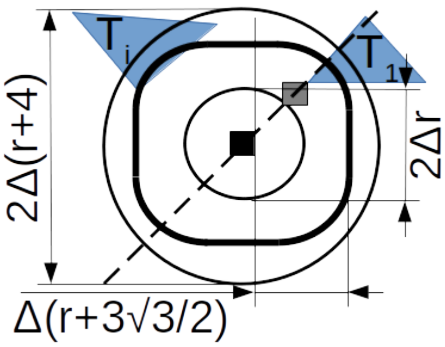

Fig. 5. Schematic showing the geometry used to prove Lemma 4. The black square is voxel V, the grey square is voxel N_1, triangles outside the rounded square are not in C_2^*, and triangles in the large circle are in L'.

Theorem 1: Distance queries using the proposed method do not test more triangles than would be tested using the Linked Voxel Structure, provided the linked voxel structure is modified for correctness.

Proof: By Lemmas 3 and 4. ■

The VCGLib suite [23] provides an implementation of a third voxelization-based distance query (referred to herein as the "VCG voxelization") that was developed for the Metro tool [17]. The triangles associated to each voxel are those triangles whose axis aligned bounding boxes intersect the voxel. At query time, these triangles have their distances to the point computed, if any. Cells adjacent to the voxel containing the query point are processed in order of increasing distance until all cells closer than the minimum distance found have been checked. The triangles tested will therefore depend on the location of the query point within the voxel, making a theoretical comparison with the proposed method challenging. The numerical results in section VI demonstrate that the proposed method is much faster in practice, in part due to avoiding an iterative process at query time.

VI. Numerical Comparisons

In addition to the proposed method, the authors implemented the RSS tree based distance query of [4], a BSP distance query following [2] and [20], and the linked voxel structure of [2]. Results for the linked voxel structure with BSP trees created in each non-empty voxel, as described

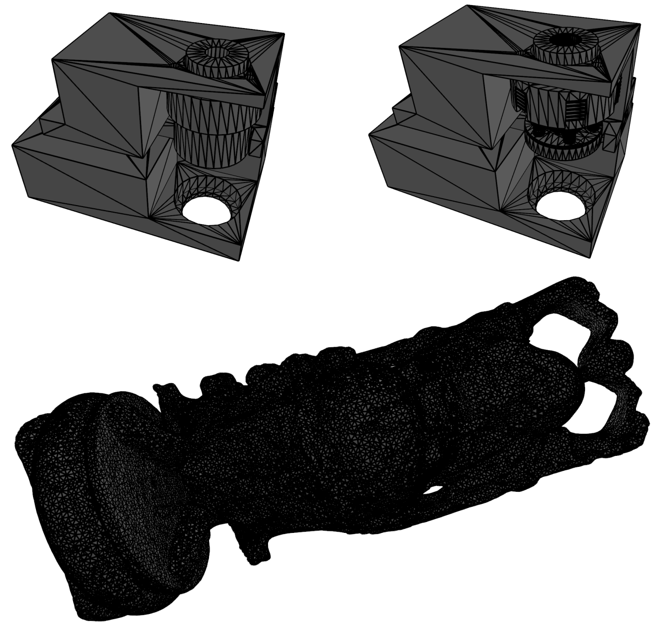
Fig. 6. Three meshes of varying properties: a "Coarse Hinge" (1,874 triangles), a "Fine Hinge" (137,136 triangles), and the "Happy Buddha" statue from the Stanford 3D Scanning Repository (67,240 triangles)

in [2], are not included as we found that for the small numbers of triangles included in each voxel there was no benefit. Comparisons were also performed with the AABB tree distance query included in the CGAL library, which uses a kd-tree of points on the mesh to get an initial upper bound for the distance to improve pruning performance [12], and the voxelization approach from the VCG library [23].

Testing was conducted using three meshes of varying triangle count and feature complexity, shown in figure 6. The "Coarse Hinge" represents a CAD model that has been manually simplified (1,874 triangles), with only a few planes and cylinders remaining. The "Fine Hinge" is the same part without simplification (137,136 triangles), and now includes detailed helices, fillets, and thin plates. Finally, the statue from the Stanford 3D Scanning Repository (67,240 triangles) exhibits roughly uniform triangle size across the mesh. The axis aligned bounding box of the hinge (which fits the hinge tightly) is 16.2 cm X 10.2 cm X 12.6cm. The statue's axis aligned bounding box is 8.1 cm X 19.8 cm X 8.1 cm.

Timing results are for a single thread on an Intel 11th Gen Core i7-11800H. The proposed method, RSS Tree, BSP Tree, and Linked Voxel techniques are implemented in Python 3.8.10 and compiled using numba 0.56.4. Timing for the linked voxel structure is reported for linking out to $\Delta(r+3)$ not $\Delta(r+4)$ as the performance degrades dramatically and the accuracy loss is negligible in practice. CGAL uses version 5.00.2.100 and the official CGAL SWIG bindings (version 5.5.1). VCGLib results are obtained using version 2022.02, compiled using gcc 9.4.0.

The voxelization techniques require a grid size, which was set to 1 cm for the coarse and fine hinge and 2 mm for the statue. Denser grids translate directly to faster query times at the cost of increased construction cost and memory footprint; these values were chosen manually to ensure acceptable performance of all three voxelization methods.

The RSS tree and BSP tree implementations allow specifying a size for the leaf nodes. These values were chosen independently for each mesh to achieve good performance. The RSS tree used 40 triangles/leaf on the coarse hinge, 50 on the fine hinge, and 60 on the statue. The BSP tree used 120 triangles/leaf on the coarse hinge and 240 triangles/leaf on both the fine hinge and the statue. In contrast, the CGAL AABB-tree implementation has no user-specified parameters as it always constructs single triangle leaf nodes. The time to construct the voxelization or tree is recorded in table I; the proposed method uses an RSS Tree and 16 thread CPU parallelism, while linked voxel construction uses an RSStree and CUDA.

Table II reports the time needed to compute the distance from the mesh to 1000 points uniformly sampled in 1.5x the axis aligned bounding box of the input mesh, once all pre-computed data structures have been created. Timing comparisons between the voxel and non-voxel methods are of limited utility as the voxel grid may always be made finer to achieve faster queries; they are reported here to show that for reasonable voxel sizes the proposed method outperforms tree based methods. Note the excellent performance of the RSS Tree technique on the Fine Hinge, which has widely varying triangle sizes. Handling this case was a particular focus of the paper from which it is derived [4].

For fixed grid sizes, as suggested by theorem 1, the proposed method tests many fewer triangles than the linked voxel structure. Note that the proposed method runs faster than the number of triangles tested would suggest; this is because it has no query time calculations beyond looking up the triangles to test and doing so. Tree based methods must traverse the tree at runtime. For the linked voxel structure, empty voxels require an extra indirection to every linked voxel and most voxels are empty. The VCGLib approach must determine at runtime which non-intersecting triangles need to be tested, but the actual number of triangles tested is not readily accessible.

A CUDA implementation of the proposed method was developed using the numba CUDA interface. 32-bit floats were used throughout the GPU implementation. The implementation was run on a GeForce RTX 3080 Laptop GPU for the same three meshes, using the same voxelizations as for the CPU results in table II. Averaged over ten executions on 1 million query points uniformly sampled in 1.5x the axis aligned bounding box of the input mesh, the coarse hinge required 9.66 ms, the fine hinge required 407 ms, and the statue required 15.7 ms. This includes time to transfer query points to the GPU and distances back to the CPU. Even for the fine hinge, the queries/second using the GPU increased by a factor of 45. For the Coarse Hinge the throughput increased by 124x, while for the statue the increase was by a factor of 462.

VII. Conclusion

The proposed method is theoretically superior to all three existing voxelization approaches considered. Numerical ex-

TABLE I

TIME TO CONSTRUCT THE ACCELERATION DATA STRUCTURE FOR EACH ALGORITHM AND MESH.

Method	Coarse Hinge	Fine Hinge	Buddha
Proposed Method	850 ms	19.2 s	242 s
Linked Voxels	2.81 s	16.6 s	3732 s
VCG Voxels	**2.83** ms	**89.2** ms	**77.1** ms
RSS Tree	37.7 ms	9.49 s	3.3 s
BSP Tree	5.23 ms	990 ms	402 ms
CGAL	13 ms	976 ms	477 ms

TABLE II

REPORTS TIME TO COMPUTE THE DISTANCE FROM 1000 RANDOM POINTS IN 1.5X THE AXIS ALIGNED BOUNDING BOX OF THE MESH, AVERAGED OVER 10 EXECUTIONS. IN PARENTHESES ARE THE AVERAGE NUMBER OF TRIANGLES TESTED PER POINT QUERIED, IF AVAILABLE.

Method	Coarse Hinge	Fine Hinge	Buddha
Proposed Method	**1.2** ms (**41**)	**18.3** ms (960)	**7.26** ms (**275**)
Linked Voxels	32.4 ms (355)	1.64 s (23,412)	124 ms (1,596)
VCG Voxels	17.6 ms	36.9 ms	657 ms
RSS Tree	13.6 ms (78)	19.7 ms (**116**)	35.4 ms (300)
BSP Tree	11.1 ms (229)	92.8 ms (1,717)	223 ms (4,502)
CGAL	19.7 ms	42.6 ms	14.1 ms

periments have demonstrated that this theoretical advantage holds in practice, achieving between 2 and 100-fold speed-ups for equal grid size, depending on the mesh. This comes at the cost of construction complexity; though not a focus of this work we found that the VCG voxelization could generally build much faster than the proposed method.

The proposed method would benefit from improvements to the construction of the voxelization to allow building denser grids on finer meshes. At a basic level, algorithmic improvements to reuse information about triangle to voxel corner distances between neighboring voxels would make a large difference at the cost of reducing the opportunity for parallelism. While the voxel-based distance query has been ported to the GPU, the construction of the grid itself has not been GPU accelerated. The use of an adaptive grid, like the octree-based approach in [22], would also be beneficial as it would mitigate the memory footprint of dense voxelizations and allow handling query points far from the mesh surface.

REFERENCES

[1] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha, “Fast distance queries with rectangular swept sphere volumes,” in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 4, 2000, pp. 3719–3726 vol.4.

[2] S. Hauth, Y. Murtezaoglu, and L. Linsen, “Extended linked voxel structure for point-to-mesh distance computation and its application to nc collision detection,” *Computer-Aided Design*, vol. 41, no. 12, pp. 896–906, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010448509001754

[3] C. Lauterbach, Q. Mo, and D. Manocha, “gproximity: Hierarchical gpu-based operations for collision and distance queries,” *Computer Graphics Forum*, vol. 29, no. 2, pp. 419–428, 2010. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2009.01611.x

[4] Y. Li, E. Shellshear, R. Bohlin, and J. S. Carlson, “Construction of bounding volume hierarchies for triangle meshes with mixed face sizes,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 9191–9195.

[5] W. Li and P. Song, “A modified icp algorithm based on dynamic adjustment factor for registration of point cloud and cad model,” *Pattern Recognition Letters*, vol. 65, pp. 88–94, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167865515002287

[6] A. Petit, V. Lippiello, and B. Siciliano, “Real-time tracking of 3d elastic objects with an rgb-d sensor,” in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 3914–3921.

[7] J. Sanchez, A. Segura, and I. Barandiaran, “Fast and accurate mesh registration applied to in-line dimensional inspection processes,” *International Journal on Interactive Design and Manufacturing*, vol. 12, pp. 877–887, August 2018.

[8] D. Mejia-Parra, J. Lalinde-Pulido, J. R. Sánchez, O. Ruiz-Salguero, J. Posada, C. Cae, and U. Eafit, “Perfect spatial hashing for point-cloud-to-mesh registration.” in *CEIG*, 2019, pp. 41–50.

[9] C. Sigg, R. Peikert, and M. Gross, “Signed distance transform using graphics hardware,” in *IEEE Visualization, 2003. VIS 2003.*, 2003, pp. 83–90.

[10] H. Zhao, “A fast sweeping method for eikonal equations,” *Mathematics of computation*, vol. 74, no. 250, pp. 603–627, 2005.

[11] P. F. Felzenszwalb and D. P. Huttenlocher, “Distance transforms of sampled functions,” *Theory of computing*, vol. 8, no. 1, pp. 415–428, 2012.

[12] P. Alliez, S. Tayeb, and C. Wormser, “3D fast intersection and distance computation,” in *CGAL User and Reference Manual*, 5.5.1 ed. CGAL Editorial Board, 2022. [Online]. Available: https://doc.cgal.org/5.5.1/Manual/packages.html#PkgAABBTree

[13] Dawson-Haggerty et al., “trimesh.” [Online]. Available: https://trimsh.org/

[14] K. E. Hoff III, J. Keyser, M. Lin, D. Manocha, and T. Culver, “Fast computation of generalized voronoi diagrams using graphics hardware,” in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 1999, pp. 277–286.

[15] Z. Yuan, G. Rong, X. Guo, and W. Wang, “Generalized voronoi diagram computation on gpu,” in *2011 Eighth International Symposium on Voronoi Diagrams in Science and Engineering*. IEEE, 2011, pp. 75–82.

[16] J. Edwards, E. Daniel, V. Pascucci, and C. Bajaj, “Approximating the generalized voronoi diagram of closely spaced objects,” in *Computer Graphics Forum*, vol. 34, no. 2. Wiley Online Library, 2015, pp. 299–309.

[17] P. Cignoni, C. Rocchini, and R. Scopigno, “Metro: measuring error on simplified surfaces,” in *Computer graphics forum*, vol. 17, no. 2. Wiley Online Library, 1998, pp. 167–174.

[18] M. Oelsch, M. Karimi, and E. Steinbach, “R-loam: Improving lidar odometry and mapping with point-to-mesh features of a known 3d reference object,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2068–2075, 2021.

[19] M. H. Sattar, “Model aware lidar odometry and mapping (ma-loam): Improving simultaneous localization and mapping accuracy by robustly leveraging a building information model,” Master’s thesis, Technische Universität München, Feb 2022.

[20] M. de Berg, M. van Krevald, M. Overmars, and O. Schwarzkopf, *Computational geometry : algorithms and applications*, 2nd ed. Berlin: Springer, 2000.

[21] M. Guthe, P. Borodin, and R. Klein, “Fast and accurate hausdorff distance calculation between meshes,” *The Journal of WSCG*, vol. 13, 2005.

[22] B. Drost and S. Ilic, “Almost constant-time 3d nearest-neighbor lookup using implicit octrees,” *Machine Vision and Applications*, vol. 29, pp. 299–311, 2 2018.

[23] “The vcg library,” 2023. [Online]. Available: http://vcg.isti.cnr.it/vcglib/

[24] Y. Kang, M.-H. Kyung, S.-H. Yoon, and M.-S. Kim, “Fast and robust hausdorff distance computation from triangle mesh to quad mesh in near-zero cases,” *Computer Aided Geometric Design*, vol. 62, pp. 91–103, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167839618300311

[25] S. Lefebvre and H. Hoppe, “Perfect spatial hashing,” *ACM Trans. Graph.*, vol. 25, no. 3, p. 579–588, jul 2006. [Online]. Available: https://doi.org/10.1145/1141911.1141926

[26] C. Ericson, *Real-time collision detection*. CRC Press, 2004.