

Efficient Proximity Queries for Continuum Robots on Parallel Computing Hardware

Konrad Leibrandt, Guang-Zhong Yang, *Fellow, IEEE*

Abstract—Surgical manipulators are increasingly capable of approaching deep seated pathologies through convoluted pathways due to advances in the field of continuum robotics. This class of robots can be, in most cases, accurately modelled as a chain of cylindrical shapes. In order to safely and seamlessly telemanipulate these robots, which have complex non-intuitive kinematics, haptic guidance schemes have been developed that rely on accurate proximity queries (PQ) to calculate the distance between the continuum robot and the anatomy. This paper introduces an approach to accurately model continuum robots using cylindrical shaped segments with spherical or flat caps and then efficiently calculate the shortest distance to a triangle mesh. Implementations of efficient, analytical narrow phase PQ calculations for simple and complex geometrical primitives suitable for parallel computing hardware are presented together with experimental validation which show improved performance suitable for real-time robotic applications. An *in silico* experiment comparing various root-finding algorithms, which are commonly used in PQ calculation or other optimization tasks are compared to assess their suitability when executed on different hardware. The implications of these experimental results are discussed, in particular with regards to the selection of a suitable proximity query algorithm depending on the available parallel computing hardware. Finally, an outlook of future improvements for dense dynamic anatomies is presented.

Index Terms—Medical Robots and Systems, Computational Geometry, Flexible Robots, Collision Avoidance, Surgical Robotics: Steerable Catheters/Needles

I. INTRODUCTION

FINDING the closest points between two objects is the objective of proximity query (PQ) algorithms. PQ algorithms have been intensively investigated in the literature and are essential to various fields, such as computer graphics [1], robot collision detection and avoidance [2], and haptics including active constraints/virtual fixtures [3]. In robotic surgery safe manipulation is dependent on accurate proximity calculations to avoid trauma and inadvertent tissue damage [4], [5]. In particular, the emerging field of surgical continuum robots, in which manipulators are designed to follow convoluted pathways to approach deep seated pathologies

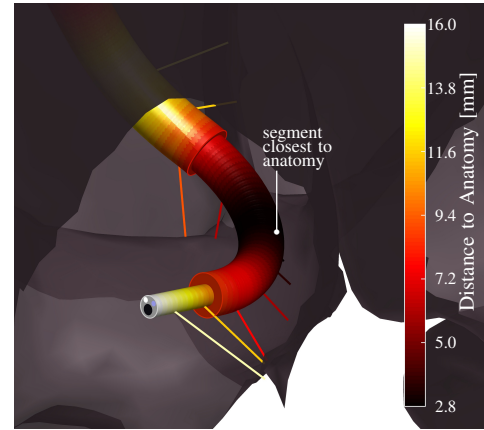


Fig. 1. Illustration of a concentric tube robot in brain ventricles featuring a light source, a camera, and a working channel at the tip. The colour indicates the distance from the robot's segments to the anatomy. The lines connect respective closest points between segments on the robot and the anatomy.

through a single incision or natural orifices, requires accurate information about proximity to the delicate anatomy since often both robot control and haptic feedback rely on it [6], [7].

To efficiently evaluate the constraints for the manipulator, spatial partitioning and bounding volume hierarchies are common approaches [8]. Their initial generation, however, is often time consuming and therefore not suitable for highly dynamic environments [5]. Alternatively, computations are divided into a broad phase, to quickly reject combinations of objects that are sufficiently far apart, and a narrow phase to accurately determine the distance of the remaining objects. Broad phase collision detection commonly uses bounding volumes such as AABB, OBB, sphere, or torus [1]. These, however, do not address the accurate distance calculation itself. Some narrow phase algorithms are based on Voroni regions [9], [10] to determine the shortest distance via an iterative process. Other algorithms determine proximity by iteratively optimizing the distances of simplices of convex shapes [11], or represent the objects as implicit surfaces [12]. Although very efficient, they are sensitive to the initial guesses and are therefore mainly used in tracking applications, where closest points between robot and environment shift mostly locally, due to continuous motion. However, some application such as sampling-based approaches used in path planning or continuum robot design, do not necessarily provide good initial guesses for the proximity calculation [13]. Recently, an optimization approach to calculate the closest distance between a triangle and a

Manuscript received: September, 9, 2016; Revised December, 6, 2016; Accepted January, 18, 2016.

This paper was recommended for publication by Editor Ken Masamune upon evaluation of the Associate Editor and Reviewers' comments.

This work is supported by the Department of Health and Wellcome Trust through the Health Innovation Challenge Fund (HICF-T4-299). K. Leibrandt was supported by the President's PhD Scholarship of Imperial College London and EPSRC UK.

All authors are with the Hamlyn Centre for Robotic Surgery, Imperial College London, London, United Kingdom. Corresponding author: k.leibrandt12@imperial.ac.uk

Digital Object Identifier (DOI): see top of this page.

generalized cylinder/cone was presented, providing realtime proximity queries for active constraint applications using stream processing on a GPU [3]. The approach iteratively minimizes the distance between a cylinder/cone shape to the triangle and uses mesh culling in a preceding broad phase step to speedup overall computation time. Analytical solutions to calculate the shortest distance between geometric primitives do not make any assumptions of the investigated structures and are not sensitive to initial guesses. Furthermore, many of the aforementioned efficient broad phase and near phase techniques use analytical solvers for very simple primitives such as spheres, lines, or segments. However, sometimes those primitives cannot accurately describe continuum robots with flat endings and changing radii, see Fig. 1. Such robotic devices are for example used for procedures to inspect and image pathologies [14], [15] or to remove tissue using suction [16]. Additional end-effectors like grippers can be modelled with standard polygon meshes.

This paper introduces a mathematically and conceptually different approach to [3] for calculating the shortest distance between a cylindrical primitive and a triangle. It contributes on efficient calculation of the shortest distances of simple geometries such as segments, by avoiding branching in the instruction sequence. Furthermore, the analytical computation of shortest distances of a triangle to different cylindrical primitives, namely a capsule with spherical ends, a cylinder with flat ends, and a hybrid of the two is presented. These primitives are used to accurately model the surface of a continuum robot. To create the model an algorithm is formalized to generate a chain of different cylinder shapes based on the robot's centreline and radii. Finally, simulations are conducted to compare the presented work to existing algorithms, including a study to assess the suitability of various root-finding algorithms used in PQ calculation and other optimization tasks, on different computing hardware.

Note the $:=$ operator denotes assignment to define helper variables, and the scalar product of two vectors $\mathbf{a}$, $\mathbf{b}$ is denoted as: $\langle \mathbf{a}, \mathbf{b} \rangle$.

II. OPTIMIZATION FOR BOUNDED PRIMITIVES

Most geometric primitives used to describe anatomy or robotic tools have parameter bounds in their description. For two line segments $\mathbf{S}$ and $\mathbf{U}$, consisting of points $\mathbf{S}_{0,1}$ and

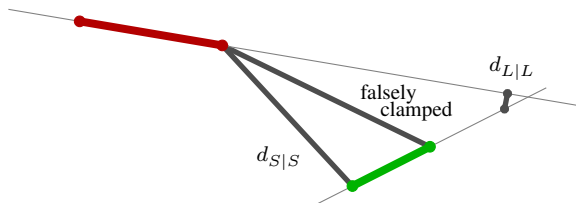


Fig. 2. Illustration of clamping for distance segment to segment calculation, $d_{L|L}$: shortest distance of lines, falsely clamped: distance when parameters of $d_{L|L}$ -calculation are simply clamped to parameter bounds (not shortest), $d_{S|S}$: shortest distance of two segments.

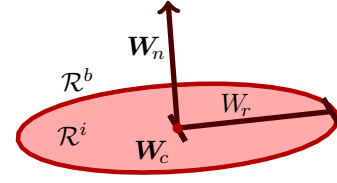


Fig. 3. Geometrical description of a disc. The disc consisting of the interior $\mathcal{R}^i$ area and the boundary circle $\mathcal{R}^b$, and is determined by the centre $\mathbf{W}_c$, the normal $\mathbf{W}_n$ of the plane it lies in, and the radius $\mathbf{W}_r$.

$\mathbf{U}_{0,1}$ respectively

$$\mathbf{S} := \mathbf{S}_0 + t (\mathbf{S}_1 - \mathbf{S}_0), t \in [0, 1] \quad (1)$$

$$\mathbf{U} := \mathbf{U}_0 + v (\mathbf{U}_1 - \mathbf{U}_0), v \in [0, 1], \quad (2)$$

the squared distance $\|\mathbf{S} - \mathbf{U}\|^2$ is known, see [17], to be calculated as,

$$\begin{aligned} \|\mathbf{S} - \mathbf{U}\|^2 = & t^2 \underbrace{\|\mathbf{S}_1 - \mathbf{S}_0\|^2}_a + 2tv \underbrace{\langle \mathbf{S}_1 - \mathbf{S}_0, \mathbf{U}_0 - \mathbf{U}_1 \rangle}_b + \\ & v^2 \underbrace{\|\mathbf{U}_0 - \mathbf{U}_1\|^2}_c + 2t \underbrace{\langle \mathbf{S}_1 - \mathbf{S}_0, \mathbf{S}_0 - \mathbf{U}_0 \rangle}_d + \\ & 2v \underbrace{\langle \mathbf{U}_0 - \mathbf{U}_1, \mathbf{S}_0 - \mathbf{U}_0 \rangle}_e + \underbrace{\|\mathbf{S}_0 - \mathbf{U}_0\|^2}_f \end{aligned} \quad (3)$$

and finding the shortest distance is equivalent to solving,

$$\nabla \|\mathbf{S} - \mathbf{U}\|^2 = \left[\frac{\partial \|\mathbf{S} - \mathbf{U}\|^2}{\partial t}, \frac{\partial \|\mathbf{S} - \mathbf{U}\|^2}{\partial v} \right]^T \stackrel{!}{=} [0, 0]^T \quad (4)$$

which is satisfied by

$$t = \frac{eb - dc}{ac - b^2}, \quad v = \frac{db - ea}{ac - b^2}, \quad (5)$$

$$\text{for } (\mathbf{S}_1 - \mathbf{S}_0) \nparallel (\mathbf{U}_1 - \mathbf{U}_0) \Rightarrow ac - b^2 > 0.$$

Since both parameters are bound to the interval $t, v \in [0, 1]$ the solution in (5) is only valid *iff* the closest points described by the parameter lie between the respective end points. A simple clamping of the two parameters using,

$$\text{clamp}(v, v_{lo}, v_{hi}) = \min \{ \max \{ v, v_{lo} \}, v_{hi} \} \quad (6)$$

to bound them to the interval $[0, 1]$ such that the lower and upper bounds are $v_{lo} = 0$, and $v_{hi} = 1$ respectively, is incorrect. The issue with clamping is illustrated in Fig. 2. Therefore, one approach is to use decision trees, to identify the correct parameters [17]. These decision trees are implemented as *if-else*-statements, and allow to postpone floating point divisions until they are required. However, modern processor architectures, in particular accelerator hardware (*e.g.* GPU, or many integrated core architecture (MIC)), are time sensitive to branching in the instruction sequence. To overcome this clamping, using (6), can often be used to avoid *if-else*-constructs such that branching is reduced. For example, the correct parameters to calculate the shortest distance between two segments, can be formulated as,

$$\hat{t} = \frac{\text{clamp}(\text{clamp}(tc, -d, -(d+b)), 0, c)}{c} \quad (7)$$

$$\hat{v} = \frac{\text{clamp}(\text{clamp}(va, -e, (b-e)), 0, a)}{a}. \quad (8)$$

This approach¹, of avoiding branching using clamping-, min-, or max-functions, can also be used for other shortest distance calculation such as point to triangle, or more simple primitives, as described in section III-A.

III. SHORTEST DISTANCE TO CONTINUUM ROBOT

Calculating the shortest distance between two segments is an important component to analytically calculate the shortest distance between a triangle and a cylinder that represents the continuum robot. The cylinder's centreline can be considered as a segment, and the triangle's edges too. The second important component is to consider the flat ends of a cylinder. The ends of a cylinder are denoted as disks, and a disk is defined by $\mathbf{W}_c$ (centre point), $\mathbf{W}_n$ (normal), W_r (radius), as illustrated in Fig. 3. In contrast to the circle $\mathcal{C} = \{\mathbf{C}_c, \mathbf{C}_n, C_r\}$ which is a curve (one dimensional primitive) the disk represents a surface (two dimensional) in $\mathbb{R}^3$. Meaning the interior of the circle is part of the disk primitive. In the following $P_{A|B}$ denotes the closest point on primitive A in a shortest distance calculation between the objects A and B . The respective shortest distance is denoted as $d_{A|B}$.

A. Point to Disc

The calculation of the shortest distance between a point $\mathbf{P}$ and a disc $\mathcal{W} = \{\mathbf{W}_c, \mathbf{W}_n, W_r\}$, is similar to calculating the distance between a point-plane, where the closest point on the plane is restricted to a distance interval of $[0, W_r]$ to the disc's centre:

$$\mathbf{D} := \mathbf{P} - \mathbf{W}_c \quad (9)$$

$$l := \langle \mathbf{D}, \mathbf{W}_n \rangle \quad (10)$$

$$\hat{\mathbf{P}}_{W|P} = \mathbf{W}_c + \min \left\{ 1, \frac{W_r}{\|\mathbf{D} - l\mathbf{W}_n\|} \right\} (\mathbf{D} - l\mathbf{W}_n) \quad (11)$$

$$\hat{d}_{W|P} = \|\mathbf{P}_{\text{con}} - \mathbf{P}\| \quad (12)$$

Where the closest point on the disc to $\mathbf{P}$ is $\hat{\mathbf{P}}_{W|P}$ and $\hat{d}_{W|P}$ the distance value.

¹Note, that the presented implementation can handle parallel segments, in case the min-, max-function returns the non-*NaN* parameter in case one value is *NaN* (e.g. *fmin*, *fmax* in C/C++). Also note, that this computation might lead to inaccurate results in case floating point cancellations occurs in (5), (7), or (8), other approaches based on optimization mitigate this issue at the cost of computational time

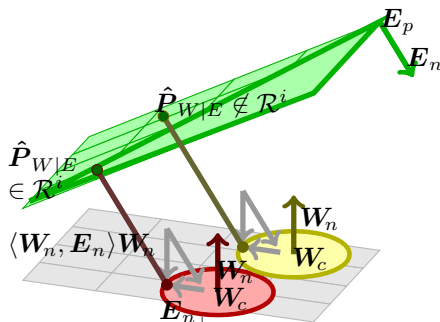


Fig. 4. Shortest distance, plane to disc, and triangle to disc. Red disk: closest point on plane lies with the triangle interior, Yellow disk: closest point on plane lies outside of the triangle.

B. Segment to Disc

The shortest distance between a line segment $\mathcal{S} = \{\mathbf{S}_0, \mathbf{S}_1\}$, with $\mathbf{S}_0, \mathbf{S}_1$ denote the end points of the segment, and a disc can be calculated by adapting the calculation approach of a line to a circle, [18]. For a point $\mathbf{P}(s)$ on a parametrized curve, the point closest to $\mathbf{P}(s)$ on the circle $\mathcal{C}$ is denoted as $\hat{\mathbf{P}}_{C|P_s}$. It can be calculated as:

$$\mathbf{D}(s) := \mathbf{P}(s) - \mathbf{C}_c \quad (13)$$

$$l(s) := \langle \mathbf{D}(s), \mathbf{C}_n \rangle \quad (14)$$

$$\hat{\mathbf{P}}_{C|P_s}(s) = \mathbf{C}_c + C_r \frac{\mathbf{D}(s) - l(s)\mathbf{C}_n}{\|\mathbf{D}(s) - l(s)\mathbf{C}_n\|} \quad (15)$$

and thus the squared distance is:

$$d_{C|P_s}(s)^2 = l(s)^2 + (\|\mathbf{C}_n \times \mathbf{D}(s)\| - C_r)^2. \quad (16)$$

Defining the curve as a segment in parameter form,

$$\mathbf{P}(s) := \mathbf{A} + \mathbf{B}s, \quad (17)$$

with $\mathbf{A} = (\mathbf{S}_0 + \mathbf{S}_1)/2$ the centre point of the segment, $\mathbf{B} = (\mathbf{S}_1 - \mathbf{S}_0)/\|\mathbf{S}_1 - \mathbf{S}_0\|$ the unit direction vector of the segment, and s is the bounded parameter within the interval $s \in [-1/2\|\mathbf{S}_1 - \mathbf{S}_0\|, 1/2\|\mathbf{S}_1 - \mathbf{S}_0\|]$. The shortest distance can be calculated by solving for the roots of the squared distance derivative ($\partial(d_{C|P_s}(s)^2)/\partial s \stackrel{!}{=} 0$), which is equivalent to finding the minimum of (16), and requires to solve for the roots of:

$$\begin{aligned} & (\|\mathbf{C}_n \times \mathbf{B}\|^2 s^2 + \|\mathbf{C}_n \times \mathbf{D}_0\|^2 + \\ & 2s(\langle \mathbf{B}, \mathbf{D}_0 \rangle - \langle \mathbf{C}_n, \mathbf{B} \rangle \langle \mathbf{C}_n, \mathbf{D}_0 \rangle)) (s + \langle \mathbf{B}, \mathbf{D}_0 \rangle)^2 = \\ & C_r^2 (\|\mathbf{C}_n \times \mathbf{B}\|^2 s + \langle \mathbf{B}, \mathbf{D}_0 \rangle - \langle \mathbf{C}_n, \mathbf{B} \rangle \langle \mathbf{C}_n, \mathbf{D}_0 \rangle)^2 \end{aligned} \quad (18)$$

which represents a quartic polynomial in s . Thus, the real roots of (18) which are in the bounds of s represent local extrema for the distance. Since the bounds of s are known the global minimum $\hat{s}$ is either at those bounds or at one of the local extrema, such that the closest point to the segment is found by substituting $\hat{s}$ into (15). The point on a segment $\hat{\mathbf{P}}_{S|W}$ closest to a disc can be calculated based on: i. the point on a segment closest to a circle $\hat{\mathbf{P}}_{S|C}$, and ii. the point on a segment closest to the plane ($\mathbf{E}_w$ plane in which the disk $\mathcal{W}$ lies) $\hat{\mathbf{P}}_{S|E_w}$. $\hat{\mathbf{P}}_{S|W}$ is calculated as,

$$\hat{\mathbf{P}}_{S|W} = \begin{cases} \hat{\mathbf{P}}_{S|E_w}, & \text{if } \|\hat{\mathbf{P}}_{E_w|S} - \mathbf{W}_c\| < W_r \\ \hat{\mathbf{P}}_{S|C}, & \text{else} \end{cases}, \quad (19)$$

C. Plane to Disc

For a plane $\mathcal{E} = \{\mathbf{E}_p, \mathbf{E}_n\}$, with $\mathbf{E}_p$ a point on the plane and $\mathbf{E}_n$ the normal of the plane. The point on the disc closest to the disc is calculated as,

$$\mathbf{E}_{n_{\perp}} := \mathbf{E}_n - \langle \mathbf{E}_n, \mathbf{W}_n \rangle \mathbf{E}_n \quad (20)$$

$$\hat{\mathbf{P}}_{W|E} = \mathbf{W}_c + W_r \frac{\text{sgn}(\langle \mathbf{W}_n, \mathbf{E}_p - \mathbf{W}_c \rangle) \mathbf{E}_{n_{\perp}}}{\|\mathbf{E}_{n_{\perp}}\|}, \quad (21)$$

illustrated in Fig. 4.

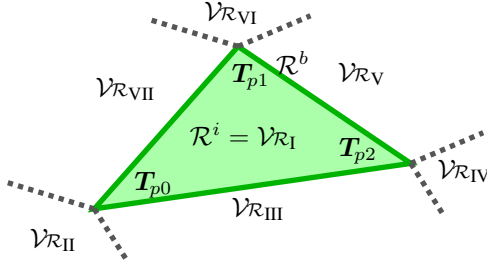


Fig. 5. Geometrical description of a triangle. The triangle consists of the interior area $\mathcal{R}^i$ and the three boundary segments $\mathcal{R}^b$, it has seven Voronoi regions $\mathcal{V}_{R_I}$ - $\mathcal{V}_{R_{VII}}$, separated by the three vertices T_{p0} - T_{p2} .

D. Triangle to Disc

To calculate the shortest distance between a triangle and a disc two cases have to be considered. The shortest distance can be between the triangle's interior $\mathcal{R}_T^i$ and $\mathcal{W}$, or the triangle's boundary $\mathcal{R}_T^b$ and $\mathcal{W}$. Constructing a plane $\mathcal{E}_t$, containing the interior $\mathcal{R}_T^i$, such that triangle $\mathcal{T} = \{T_{p0}, T_{p1}, T_{p2}\}$, see Fig. 5, lies in it,

$$\mathbf{E}_{p_t} = T_{p0}, \mathbf{E}_{n_t} = \frac{(T_{p1} - T_{p0}) \times (T_{p2} - T_{p0})}{\|(T_{p1} - T_{p0}) \times (T_{p2} - T_{p0})\|}, \quad (22)$$

and defining the boundary $\mathcal{R}_T^b$ as three segments,

$$\mathcal{S}_{t0} = \{T_{p0}, T_{p1}\}, \mathcal{S}_{t1} = \{T_{p0}, T_{p2}\}, \mathcal{S}_{t2} = \{T_{p2}, T_{p1}\}, \quad (23)$$

so that the shortest distance between $\mathcal{R}_T^b$ and the $\mathcal{W}$ is calculated as:

$$d_{\mathcal{R}_T^b|\mathcal{W}} = \min\{d_{\mathcal{S}_{t0}|\mathcal{W}}, d_{\mathcal{S}_{t1}|\mathcal{W}}, d_{\mathcal{S}_{t2}|\mathcal{W}}\}, \quad (24)$$

using (19) in combination with (21) yields the shortest distance between a disk and a triangle:

$$d_{T|\mathcal{W}} = \begin{cases} d_{\mathcal{E}_t|\mathcal{W}}, & \text{if } \hat{\mathbf{P}}_{\mathcal{E}_t|\mathcal{W}} \in \mathcal{R}_T^i \\ d_{\mathcal{R}_T^b|\mathcal{W}}, & \text{else} \end{cases} \quad (25)$$

E. Triangle to Capsule and Cylinder

The shortest distance of a triangle $\mathcal{T}$ to a capsule $\mathcal{K} = \{K_{p0}, K_{p1}, K_r\}$, see Fig. 6b, is calculated based on the shortest distance of a segment to a triangle $d_{S|T}$, see [17] for a mathematical and algorithmic description. Defining a segment as $\mathcal{S}_K = \{K_{p0}, K_{p1}\}$ the point on $\mathcal{K}$ closest to $\mathcal{T}$ denoted as $\hat{\mathbf{P}}_{K|T}$, is calculated as:

$$\hat{\mathbf{P}}_{K|T} = \hat{\mathbf{P}}_{\mathcal{S}_K|T} + K_r \frac{\hat{\mathbf{P}}_{T|\mathcal{S}_K} - \hat{\mathbf{P}}_{\mathcal{S}_K|T}}{\|\hat{\mathbf{P}}_{T|\mathcal{S}_K} - \hat{\mathbf{P}}_{\mathcal{S}_K|T}\|}, \quad (26)$$

which means that the distance is calculated as:

$$d_{K|T} = d_{\mathcal{S}_K|T} - K_r. \quad (27)$$

In contrast a cylinder $\mathcal{Z} = \{Z_{p0}, Z_{p1}, Z_r\}$ with flat caps, see Fig. 6a, requires to consider the previously derived distance measures between a disc $\mathcal{W}$ in case the closest point on the segment $\mathcal{S}_Z = \{Z_{p0}, Z_{p1}\}$ to the triangle is identical to one of the end points of the segment. In that case the shortest distance has to be calculated as a disk $\mathcal{W}_Z$ to triangle $\mathcal{T}$ problem. In conclusion the distance between a triangle and a cylinder is calculated as

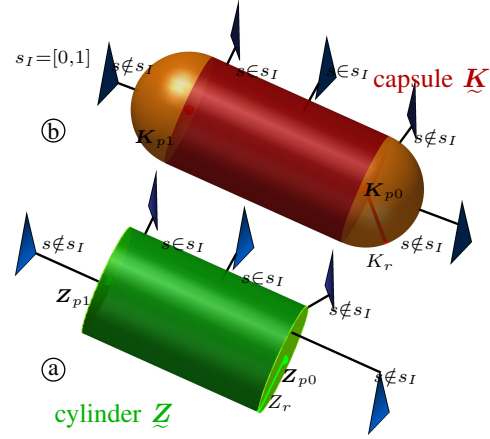


Fig. 6. Shortest distances, a: triangle to cylinder (green), and b: triangle to capsule (red). In case the line parameter s of the capsule's or cylinder's centre line is not within the interval $[0, 1]$ the closest point on the capsule or cylinder lies on the respective cap. Otherwise it lies along the linear section.

$$d_{\mathcal{S}_Z|T} = \hat{\mathbf{P}}_{T|\mathcal{S}_Z} - \hat{\mathbf{P}}_{\mathcal{S}_Z|T} \quad (28)$$

$$\hat{\mathbf{P}}_{Z|T} = \begin{cases} \hat{\mathbf{P}}_{W_{Z_0}|T} & , \text{if } \hat{\mathbf{P}}_{\mathcal{S}_Z|T} = Z_{p0} \\ \hat{\mathbf{P}}_{W_{Z_1}|T} & , \text{if } \hat{\mathbf{P}}_{\mathcal{S}_Z|T} = Z_{p1} \\ \hat{\mathbf{P}}_{\mathcal{S}_Z|T} + Z_r \frac{d_{\mathcal{S}_Z|T}}{\|d_{\mathcal{S}_Z|T}\|} & , \text{else} \end{cases} \quad (29)$$

where the respective disks in (29) are defined as

$$\mathcal{W}_{Z_0} = \{Z_{p0}, \frac{Z_{p0} - Z_{p1}}{\|Z_{p0} - Z_{p1}\|}, Z_r\}, \text{ and} \quad (30)$$

$$\mathcal{W}_{Z_1} = \{Z_{p1}, \frac{Z_{p1} - Z_{p0}}{\|Z_{p1} - Z_{p0}\|}, Z_r\}. \quad (31)$$

The respective shortest distance is calculated as the euclidean distance between the respective closest points on the triangle and cylinder,

$$d_{Z|T} = \|\hat{\mathbf{P}}_{Z|T} - \hat{\mathbf{P}}_{T|Z}\|, \quad (32)$$

where $\hat{\mathbf{P}}_{T|Z}$ is the respective closest point on the triangle which is calculated while evaluating (29), and follows the three different cases such that it lies on one of $\{\hat{\mathbf{P}}_{T|W_{Z_0}}, \hat{\mathbf{P}}_{T|W_{Z_1}}, \hat{\mathbf{P}}_{T|\mathcal{S}_Z}\}$.

F. Continuum Robot to Triangle

Continuum robots can be described as a chain of cylinders, where the centre line of that series describes a polygonal curve in $\mathbb{R}^3$. However, a simple cylinder description is not ideal for continuum robot as depicted in Fig. 7a since it results in a gap at the interface of two cylinders. This can be avoided if capsules are used, see Fig. 7b, at the disadvantage of spherical caps at the ends. This, however can be avoided, if the two end segments of the robot are instead described as hybrid, where one end is flat and the other spherical, see Fig. 7e; which however is not suitable if the distance between segments is smaller than the radius, see Fig. 7c, or if the radius is changing, see Fig. 7d.

An alternative approach is suggested in the following, which employs a forward and a backward iteration over the segments to determine if a spherical cap at the beginning or end of each segment is suitable. The approach is presented in Algorithm 1.

Its utilization results into surfaces for continuum robots as depicted in Fig. 7f-g, and it can handle varying radii along the robot centreline, see Fig. 8.

In order to use the presented hybrid segments in a proximity query, a modified cylinder description is required. Each segment is described as $\mathcal{Z}^* = \{\mathcal{Z}, S_B, S_F\}$, in which S_B describes a backwards facing sphere at $\mathbf{Z}_{p0}$, and S_F a forward facing sphere at $\mathbf{Z}_{p1}$. Furthermore, this results in an adjusted closest point calculation:

$$\hat{\mathbf{P}}_{\mathcal{Z}^*|T} = \begin{cases} \hat{\mathbf{P}}_{W_{\mathcal{Z}_0^*}|T} & , \text{if } \hat{\mathbf{P}}_{S_{\mathcal{Z}^*}|T} = \mathbf{Z}_{P_0}^* \wedge \neg S_B \\ \hat{\mathbf{P}}_{W_{\mathcal{Z}_1^*}|T} & , \text{if } \hat{\mathbf{P}}_{S_{\mathcal{Z}^*}|T} = \mathbf{Z}_{P_1}^* \wedge \neg S_F \\ \hat{\mathbf{P}}_{S_{\mathcal{Z}^*}|T} + \mathbf{Z}_r \frac{\mathbf{d}_{S_{\mathcal{Z}^*}|T}}{\|\mathbf{d}_{S_{\mathcal{Z}^*}|T}\|} & , \text{else} \end{cases} \quad (33)$$

IV. ROOTS OF POLYNOMIALS

As shown in the previous section, solving for roots of polynomials is a common problem in optimization tasks.

As presented in (18) solving the roots of a quartic polynomial is required, to calculate the shortest distance between a line, or segment to a circle. Similarly, polynomial root solving is required for calculating the shortest distance between circles in $\mathbb{R}^3$ or the overlap area of ellipses in $\mathbb{R}^2$, [19]. It is a well investigated problem, whose solution is crucial to many robotic applications from proximity queries, to imaging [20], and path planning [19].

There are closed form solutions for polynomials of degree one to four. However, due to limited numerical precision of floating point numbers, the result based on closed form computation can severely deviate from the correct solution. Although there are implementations for more robust closed form root finding, *e.g.* for quadratic polynomials:

$$f(x) = a_2 x^2 + a_1 x + a_0 \quad (34)$$

$$q = -\frac{1}{2} \left(a_1 + \text{sgn}(a_1) \sqrt{a_1^2 - 4 a_2 a_0} \right) \quad (35)$$

$$x_+ = \frac{a_0}{q}, \quad x_- = \frac{q}{a_2}, \quad (36)$$

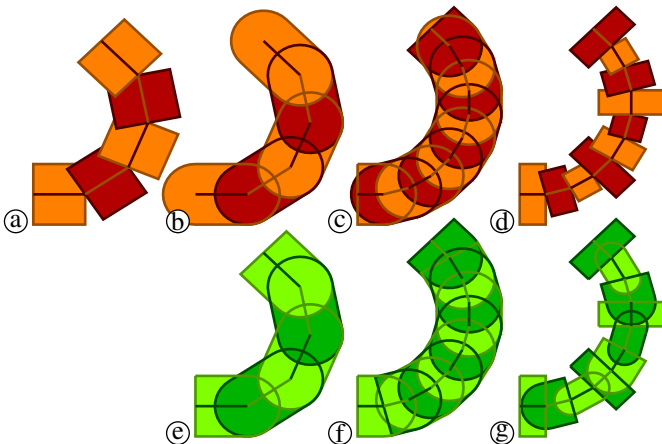


Fig. 7. Possible surface descriptions of continuum robots, assuming the end caps of the robot are flat; red: unsuitable, green: adequate.

Input: vertices on polygonal curve P , radius at each vertex R , number of vertices $\mathcal{N}_P$.

Output: boolean list l_B, l_F specifying vertices with a backward, and forward facing sphere, respectively.

```

 $P_{\text{lim}} \leftarrow P(1)$ 
for  $i \leftarrow 2$  to  $\mathcal{N}_P$  do
  if  $\lfloor \frac{\|P_{\text{lim}} - P(i)\|}{R(i)} \rfloor \geq 1 \wedge \frac{R(i)}{R(i-1)} \leq 1$  then
     $l_B(i) \leftarrow \text{True}$ 
  else if  $\frac{R(i)}{R(i-1)} > 1$  then
     $P_{\text{lim}} \leftarrow P(i)$ 
  end
end
 $P_{\text{lim}} \leftarrow P(\mathcal{N}_P)$ 
for  $i \leftarrow (\mathcal{N}_P - 1)$  to 1 do
  if  $\lfloor \frac{\|P_{\text{lim}} - P(i)\|}{R(i)} \rfloor \geq 1 \wedge \frac{R(i)}{R(i+1)} \leq 1$  then
     $l_F(i) \leftarrow \text{True}$ 
  else if  $\frac{R(i)}{R(i+1)} > 1$  then
     $P_{\text{lim}} \leftarrow P(i)$ 
  end
end
return  $l_B, l_F$ 

```

Algorithm 1: Calculating the half spheres facing backward l_B , and forward l_F .

a common alternative approach is to use iterative algorithms to find roots. Those approaches are more general and can determine roots of polynomials of degree higher than four.

Common numerical and algebra software solvers use the Jenkins-Traub, or companion matrix method, furthermore the polynomials used here have real coefficients which makes the Bairstow's method a suitable candidate. The Laguerre method is also commonly used, but not further investigated here. Properties of selected algorithms are listed in Table I. Furthermore, experiments were conducted to evaluate how different root solving algorithms perform on parallel computing hardware. Where the roots of multiple polynomials are solved in parallel, using one of the mentioned serial algorithms, the results are reported in Sec. V-C.

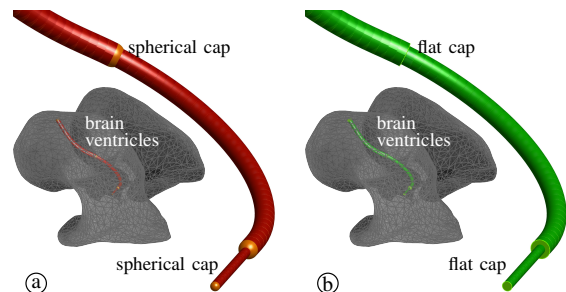


Fig. 8. Surfaces of concentric tube robots, in hydrocephalic brain ventricles, represented as polygon (triangle) mesh. a: using the capsule $\mathcal{K}$ primitive, b: using modified cylinder $\mathcal{Z}^*$ primitive.

TABLE I
POLYNOMIAL ROOTFINDING METHODS

Method	Software	Complexity	Convergence
Bairstow	-	$\mathcal{O}(n)$	local quadratic
Companion Matrix	Matlab	$\mathcal{O}(n^2)$	depending on decomposition method
Jenkins-Traub	Mathematica	$\mathcal{O}(n)$	quadratic (slightly faster)

V. EXPERIMENTAL RESULTS

A. Branching Optimized Distance Measures

The reduction of branching in distance calculations is based on clamping of the respective parameters of the investigated primitive. The benchmarks are performed using pseudo-random data, its generator is seeded in each benchmark identical, to ensure comparability.

a) *Segment to Segment*: The first benchmark highlights the computational improvements achieved when removing all *if-else*-statements and replace them by clamping as presented in (7), (8). Clamping does not result in branching, which can be assessed by inspecting the produced assembly code. Benchmark results are listed in Table II. Experiments were conducted on a CPU (Intel® Core™ i7-3770, single thread, C++ implementation), a MIC (Intel® Xeon Phi™ 7120P, OpenCL™ implementation), and a GPU (AMD FirePro™ W5000, OpenCL™ implementation). The results are listed in Table II. On the CPU the improvements are most pronounced when using 80bit floating point numbers, which might be due to the reduced instruction set for 80bit floating point numbers. The parallel computing hardware shows larger improvements, in particular the GPU hardware suffers from branching since shader units of one compute unit have to process the instructions, which can result that shader units have to wait when branching occurs.

b) *Point to Triangle*: Results for the point to triangle calculations show greater improvements on the CPU than the segment to segment calculation. However, on the GPU the improvements were marginal. *If-else*-statements were not fully removed but only reduced, which might explain why the GPU did not benefit much from implementation alterations.

TABLE II
BRANCHING OPTIMIZATION: SEGMENT TO SEGMENT

Number of PQ Calculations	T_{clamp} [ms]	T_{branch} [ms]	Speedup $\left(\frac{T_{\text{branch}}}{T_{\text{clamp}}}\right)$
CPU: i7-3770, single threaded			
float (32bit, x86)			
1×10^4 CPU	0.75	0.81	1.08
1×10^8 CPU	7135.75	7668.35	1.07
double (64bit, x86)			
1×10^4 CPU	0.90	0.93	1.04
1×10^8 CPU	8581.52	9203.24	1.07
long double (80bit, x87)			
1×10^4 CPU	2.41	3.13	1.30
1×10^8 CPU	23669.90	30391.70	1.28
GPU: W5000, MIC: 7120P, multi threaded[†]			
float (32bit, x86)			
2.4×10^5 GPU	8.3	13.0	1.57
2.4×10^5 MIC	31.1	35.7	1.15

[†]timings including data transfer to the accelerator (write/read).

TABLE III
BRANCHING OPTIMIZATION: VERTEX TO TRIANGLE

Number of PQ Calculations	T_{clamp} [ms]	T_{branch} [ms]	Speedup $\left(\frac{T_{\text{branch}}}{T_{\text{clamp}}}\right)$
CPU: i7-3770, single threaded			
float (32bit, x86)			
1×10^4	0.26	0.33	1.29
1×10^8	2303.86	2839.67	1.23
double (64bit, x86)			
1×10^4	0.27	0.31	1.15
1×10^8	2367.61	3008.59	1.27
long double (80bit, x87)			
1×10^4	0.43	1.15	2.68
1×10^8	3874.16	10494.80	2.71

B. Shortest Distance Continuum Robot to Triangles

The following benchmarks are based on realistic data sets. The triangle mesh is based on a segmented computed tomography (CT) scan of a pig's brain ventricle. The original mesh consisted of 6000 triangles, which were subdivided, maintaining the surface, and their sequence was randomized to generate a high number of unorganized triangles for the following benchmark. Cylinder/capsule segments are based on a kinematic solution of a concentric tube robot (CTR). For all benchmarks 30 segments are used, to be able to compare the results to [3]. Since this paper focuses on the analytical solution to calculate the shortest distance between primitives, no broad phase approaches are benchmarked. All benchmarks are based on exhaustive calculations to determine the shortest distance between all primitive pairs. The implementations are all based on OpenCL™ and the employed hardware is a dual-socket server CPU: 2xIntel® Xeon® E5-2637v3 (2x8 threads), and a single GPU: 1xAMD FirePro™ W9100 (2816 shader units).

a) *Cylinders $\mathcal{Z}$ to Triangles $\mathcal{T}$* : In this benchmark the segments of the CTR are all modelled as cylinders $\mathcal{Z}$, meaning that an expensive triangle to disc calculation (25) has to be computed, which involves calculating the roots of a quartic polynomial. The root finding, in the presented benchmark is based on a closed form solution implementation which gave the fastest results. Fig. 9a, shows the computational times for different numbers of triangles.

b) *Modified Cylinders $\mathcal{Z}^*$ to Triangles $\mathcal{T}$* : In a second benchmark the modified cylinders are considered and distances are calculated according to (33). Since the CTR consists of three sections four flat discs have to be considered (proximal end, two along the centreline, and tip), see Fig 8b. All inner sections are considered as capsules. In this benchmark employing the companion matrix approach for the root finding showed best results. The timing results are depicted in Fig. 9b.

c) *Capsules $\mathcal{K}$ to Triangles $\mathcal{T}$* : The last proximity query benchmark describes the CTR as multiple capsules. The results are depicted in Fig. 9c.

d) *Discussion*: The benchmarks show that the cylinder description provides the slowest computation times, which can be attributed to necessary computationally intensive root finding. Furthermore, it shows that the computation advantage using 32bit floating point in comparison to 64bit floating point is less pronounced and the GPU loses some of its advantage in

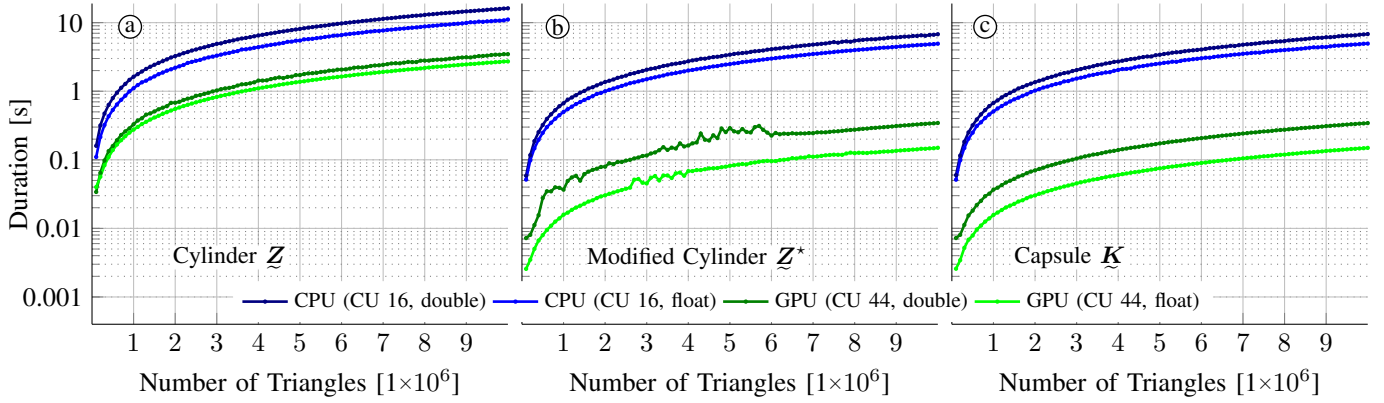


Fig. 9. Exhaustive proximity query, using 30 segments for the robot description: Primitives: a. cylinders $\mathcal{Z}$, triangles $\mathcal{T}$, b. modified cylinders $\mathcal{Z}^*$, triangles $\mathcal{T}$, c. capsules $\mathcal{K}$, triangles $\mathcal{T}$. CPU: 2xIntel® Xeon® E5-2637v3 (2x8 threads), GPU: 1xAMD FirePro™ W9100 (2816 shader units).

comparison to the other two benchmarks. When compared to the $\mathcal{K}$ benchmark using $\mathcal{Z}$ the CPU is in average 2.29x slower and the GPU 13.95x slower. Using the modified cylinder description $\mathcal{Z}^*$ only little computation speed is forfeited in comparison to $\mathcal{K}$ the CPU is as fast, and the GPU is 1.10x slower in average.

When compared to benchmarks in [3], which uses a different primitive description, to model robotic manipulators, denoted as $\mathbf{P}_{\text{comp}}$ a significant speed advantage of the approach presented here is apparent. The computational duration and a normalized performance indicator, to account for differences in the GPU, show the efficiency of the presented shortest distance calculation for $\mathcal{Z}^*$, see Table IV. Although, [3] uses a more general description, the presented primitive $\mathcal{Z}^*$ suits many continuum robots very well and might be even more accurate than [3], which omits to consider the cap at the end points of the robot. The normalized performance index Table IV shows for all primitives presented in this paper a lower computation time even when considering the very expensive $\mathcal{Z}$ primitive which requires very often root finding. When using the proposed modified cylinder primitive $\mathcal{Z}^*$ computation times are reduced by a factor of more than 36, using a CTR with three different tube radii. Distance calculations were performed with meshes consisting of $\sim 10^4$ triangles, to match those typically used in surgery [3], [13]. The corresponding GPU computation time required for 30 robot segments was 0.22 ms (double) and 0.09 ms (float), making it suitable for real-time application such as haptics or robot control.

TABLE IV
EXHAUSTIVE CALCULATION OF 30 ROBOT SEGMENTS TO
 10^6 TRIANGLE MESH COMPARED TO [3]

Primitive	T [s]	Norm. Perf. [†] [s ⁻¹ TFLOPS ⁻¹]	Comparison [‡] to $\mathbf{P}_{\text{comp}}$
$\mathcal{Z}$ (float)	0.276	0.69	2.09
$\mathcal{Z}$ (double)	0.330	1.16	3.52
$\mathcal{Z}^*$ (float)	0.016	11.93	36.15
$\mathcal{Z}^*$ (double)	0.037	10.32	31.27
$\mathcal{K}$ (float)	0.016	11.93	36.15
$\mathcal{K}$ (double)	0.037	10.32	31.27
$\mathbf{P}_{\text{comp}}$, [3]	0.946	0.33	1.00

[†] Normalized performance: Reciprocal of the product of time and GPU performance. Higher is better.

Primitives $\mathcal{Z}$, $\mathcal{Z}^*$, $\mathcal{K}$ with: 5.24 TFLOPS (float), 2.62 TFLOPS (double) (AMD FirePro W9100),

Primitive $\mathbf{P}_{\text{comp}}$ with: 3.21 TFLOPS (float) (NVIDIA GTX 770).

[‡] Ratio between normalized performance of timing of all primitives with reference to $\mathbf{P}_{\text{comp}}$.

C. Polynomial Roots

The final benchmark evaluates different approaches to find roots of polynomials with real coefficients, see Table V. During each benchmark 1×10^6 random real polynomials with coefficients in the range $[-100, 100]$ were constructed using an equally seeded pseudo-random number generator. The algorithms were all implemented in OpenCL™ and executed on a CPU (2xIntel® Xeon® E5-2637v3) and a GPU (1xAMD FirePro™ W9100). Where each compute element is tasked to find the roots of one polynomial, meaning a non-parallelized algorithm was used for an individual polynomial.

The results in Table V show that the closed form solution is particularly fast on the CPU. On the GPU it is still the fastest, however compared to the CPU the difference to the other algorithms is less distinct. Bairstow's method performs overall very fast on both hardware, however it is sensitive to initial guesses, which might be problematic for some applications. The companion matrix approach is the slowest on the CPU, which is expected since its complexity is $\mathcal{O}(n^2)$, see Table I. However, on the GPU the companion matrix is performing faster than Jenkins-Traub, which uses in its implementation many conditional statements, and is hence unsuitable for GPUs due to resulting branching. In essence, the benchmark shows that the Jenkins-Traub performs faster on the CPU and the companion matrix method performs faster on the GPU, for polynomials of degree smaller 16. However, with very high degree polynomials the companion matrix will potentially also perform worse on the GPU.

VI. DISCUSSION

The experiments in section V illustrate that the reduction of conditional expressions are particularly advantageous for GPUs. This paper showed that the efficiency of an algorithm can depend on the computing hardware. The accurate description of continuum robots allows the reduction of unnecessarily high safety margins and thus could increase the safely reachable workspace of robots, while providing high-performance proximity computation. This enables real-time collision avoidance for haptic or robotic applications. The range of applications could be further increased by combining the presented work with broadphase [1] approaches or by

TABLE V
COMPUTING TIME OF ROOT FINDING ALGORITHMS
FOR 10^6 RANDOM POLYNOMIALS

Algo- rithm	Deg- ree	CPU		GPU	
		float	double	float	double
Closed Form	2	8.70	12.82	2.65	4.07
Bairstow	2	8.58	13.35	2.59	4.11
Companion M.	2	9.08	14.71	3.04	4.66
Jenkins-Traub	2	8.56	14.67	2.66	4.32
Closed Form	3	15.44	32.54	2.21	15.44
Bairstow	3	44.80	53.30	3.19	11.77
Companion M.	3	70.44	115.71	6.53	26.18
Jenkins-Traub	3	78.82	112.19	24.66	51.11
Closed Form	4	31.83	38.27	4.35	17.25
Bairstow	4	69.67	114.40	5.91	22.51
Companion M.	4	200.54	270.40	14.20	69.78
Jenkins-Traub	4	153.04	185.10	76.12	163.36
Bairstow	6	1.79×10^2	2.95×10^2	0.12×10^2	0.27×10^2
Companion M.	6	4.78×10^2	7.58×10^2	0.47×10^2	1.67×10^2
Jenkins-Traub	6	3.57×10^2	4.41×10^2	2.22×10^2	5.37×10^2
Bairstow	8	3.12×10^2	3.97×10^2	0.34×10^2	0.97×10^2
Companion M.	8	9.36×10^2	15.34×10^2	1.67×10^2	3.48×10^2
Jenkins-Traub	8	6.65×10^2	8.26×10^2	4.67×10^2	9.86×10^2
Bairstow	10	4.30×10^2	6.63×10^2	0.53×10^2	1.35×10^2
Companion M.	10	16.04×10^2	24.30×10^2	2.81×10^2	6.53×10^2
Jenkins-Traub	10	9.55×10^2	11.34×10^2	8.02×10^2	18.17×10^2
Bairstow	15	11.34×10^2	11.90×10^2	1.73×10^2	4.26×10^2
Companion M.	15	37.52×10^2	58.67×10^2	16.31×10^2	25.98×10^2
Jenkins-Traub	15	19.68×10^2	23.21×10^2	21.05×10^2	42.30×10^2

Note: i. OpenCL™ implementation for a. Closed Form is based on GNU Scientific Library (GSL), adapted to improve robustness, b. Bairstow's method is a custom implementation, c. Companion matrix is based on GSL, adapted to avoid dynamic memory allocation, d. Jenkins-Traub is based on RPOLY of the Fortran TOMS497 library, adapted to execute on the GPU, ii. OpenCL™ results are tested against C or C++ implementations, to validate correctness, iii. Roots that are found are validated to be correct, iv. Whether all roots are found depends on the respective algorithm and not on the hardware, please refer to the extensive literature, evaluating that criteria.

employing knowledge about continuous and velocity limited motion [11] of continuum robots.

VII. CONCLUSION

This paper presents efficient proximity queries for continuum robots facilitating parallel computing hardware such as multi and many core CPUs as well as GPUs. Code optimization to speedup computation for PQ calculations by reducing branching using clamping operations are presented and evaluated for calculating the shortest distance between two line segments. Furthermore, an analytical approach to calculate the shortest distance between a triangle and different cylindrical primitives, namely cylinder, capsule, and a hybrid, modified cylinder are formalized and benchmarked. In particular, a short algorithm to generate a chain of modified cylinders based on the robots centreline and radii is presented and used to model a realistic surgical continuum robot. In comparison to other approaches from the literature, a significant speed up was measured. Finally, a study on different polynomial solvers is presented, which demonstrated that speed of a particular algorithm is dependent on the hardware type. Further work will focus on optimizing the root finding algorithm's implementation for the GPU since they slow down computation greatly. Furthermore, incorporating existing broad phase algorithms to avoid exhaustive searches will be investigated to enable proximity queries for dense dynamic anatomies within a few milliseconds.

ACKNOWLEDGMENT

The authors gratefully acknowledge Prof. Pierre Dupont from the Paediatric Cardiac Bioengineering Lab of Boston Children's Hospital, Harvard Medical School for providing the anatomical data used in this paper, and Dr. Christos Bergeles from the Translational Imaging Group, Centre for Medical Image Computing, University College London for providing the forward kinematic solver for the concentric tube robot used in the experiments and for helpful remarks towards this work.

REFERENCES

- [1] X. Zhang and Y. J. Kim, "Interactive collision detection for deformable models using streaming AABBs," *IEEE Trans. Visualization and Computer Graphics*, vol. 13, no. 2, pp. 318–329, 2007.
- [2] C. Ericson, *Real-Time Collision Detection*. Boca Raton, FL, USA: CRC Press, Inc., 2004.
- [3] K. H. Lee *et al.*, "GPU-based proximity query processing on unstructured triangular mesh model," *IEEE Int. Conf. on Robotics and Automation*, pp. 4405–4411, 2015.
- [4] K. Leibrandt *et al.*, "Implicit active constraints for a compliant surgical manipulator," in *IEEE Int. Conf. on Robotics and Automation*, 2014, pp. 276–283.
- [5] S. A. Bowyer, B. L. Davies, and F. R. y Baena, "Active constraints/virtual fixtures: A survey," *IEEE Trans. Robotics*, vol. 30, no. 1, pp. 138–157, 2014.
- [6] K. Leibrandt, C. Bergeles, and G.-Z. Yang, "On-line collision-free inverse kinematics with frictional active constraints for effective control of unstable concentric tube robots," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2015, pp. 3797–3804.
- [7] K. Leibrandt, C. Bergeles, and G.-Z. Yang, "Implicit active constraints for safe and effective guidance of unstable concentric tube robots," *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 1157–1163, 2016.
- [8] J. Williams, R. Taylor, and L. Wolff, "Augmented kd techniques for accelerated registration and distance measurement of surfaces," *Computer Aided Surgery: Computer-Integrated Surgery of the Head and Spine*, pp. 1–21, 1997.
- [9] M. C. Lin and J. F. Canny, "A fast algorithm for incremental distance calculation," in *IEEE Int. Conf. on Robotics and Automation*, 1991, pp. 1008–1014.
- [10] B. Mirtich, "V-clip: Fast and robust polyhedral collision detection," *ACM Trans. Graph.*, vol. 17, no. 3, pp. 177–208, 1998.
- [11] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal on Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.
- [12] N. Chakraborty *et al.*, "Proximity queries between convex objects: An interior point approach for implicit surfaces," *IEEE Trans. on Robotics*, vol. 24, no. 1, pp. 211–220, 2008.
- [13] C. Bergeles *et al.*, "Concentric tube robot design and optimization based on task and anatomical constraints," *IEEE Trans. Robotics*, vol. 31, no. 1, pp. 67–84, 2015.
- [14] A. Degani *et al.*, "Highly articulated robotic probe for minimally invasive surgery," *IEEE Int. Conf. on Robotics and Automation*, pp. 4167–4172, 2006.
- [15] C. J. Payne *et al.*, "A hand-held flexible mechatronic device for arthroscopy," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2015, pp. 817–823.
- [16] J. Burgner *et al.*, "Debulking from within: a robotic steerable cannula for intracerebral hemorrhage evacuation," *IEEE Trans. Biomedical Engineering*, vol. 60, no. 9, pp. 2567–2575, 2013.
- [17] P. J. Schneider and D. Eberly, *Geometric Tools for Computer Graphics*. New York, NY, USA: Elsevier Science Inc., 2002.
- [18] D. H. Eberly, *3D Game Engine Design, Second Edition: A Practical Approach to Real-Time Computer Graphics*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.
- [19] G. B. Hughes and M. Chraibi, "Calculating ellipse overlap areas," *Computing and Visualization in Science*, vol. 15, no. 5, pp. 291–301, 2012.
- [20] S. Giannarou *et al.*, "Vision-based deformation recovery for intraoperative force estimation of tool-tissue interaction for neurosurgery," *Int. Journal of Computer Assisted Radiology and Surgery*, pp. 1–8, 2016.