Acceleration of Real-time Proximity Query for Dynamic Active Constraints

Thomas C.P. Chau[1], Ka-Wai Kwok[1,3,4], Gary C.T. Chow[1], Kuen Hung Tsoi[1], Kit-Hang Lee[3], Zion Tse[4], Peter Y.K. Cheung[2], Wayne Luk[1]
[1]Department of Computing, Imperial College London, UK
[2]Department of Electrical and Electronic Engineering, Imperial College London, UK
[3]Department of Mechanical Engineering, The University of Hong Kong, Hong Kong SAR
[4]College of Engineering, The University of Georgia, Athens, GA, USA
Email: {c.chau10,k.kwok07,c.chow07,k.tsoi,p.cheung,w.luk}@imperial.ac.uk
{kwokkw,brianlkh}@hku.hk, ziontse@uga.edu

***Abstract*—Proximity Query (PQ) is a process to calculate the relative placement of objects. It is a critical task for many applications such as robot motion planning, but it is often too computationally demanding for real-time applications, particularly those involving human-robot collaborative control. This paper derives a PQ formulation which can support non-convex objects represented by meshes or cloud points. We optimise the proposed PQ for reconfigurable hardware by function transformation and reduced precision, resulting in a novel data structure and memory architecture for data streaming while maintaining the accuracy of results. Run-time reconfiguration is adopted for dynamic precision optimisation. Experimental results show that our optimised PQ implementation on a reconfigurable platform with four FPGAs is 58 times faster than an optimised CPU implementation with 12 cores, 9 times faster than a GPU, and 3 times faster than a double precision implementation with four FPGAs.**

I. Introduction

Advanced surgical robots support image guidance and haptic (force-based) feedback for effective navigation of surgical instruments. Such image-guided robots rely on computing in real-time the intersection or the closest point-pair between two objects in three-dimensional space; this computation is known as Proximity Query (PQ).

PQ has been widely studied in areas such as robot motion planning, haptics rendering, virtual prototyping, computer graphics, and animation [1]. Robot motion planning is particularly demanding for the real-time performance of PQ [2]. In the past decade, PQ has also been used as a key task for Active Constraints [3] or Virtual Fixtures [4], a collaborative control strategy mostly applied in image-guided surgical robotics. The clinical potential of this control strategy has been demonstrated by imposing haptic feedback [5] on instrument manipulation based on imaging data [6]. This haptic feedback provides the operator with kinaesthetic perception for sensing positions, velocities, forces, constraints and inertia associated with direct maneuvering of surgical instrument within the target anatomy.

As mentioned above, fast and efficient PQ is a pre-requisite for effective navigation through access routes to the target anatomy [3]. This haptic guidance, rendered based on imaging data, can enable a distinct awareness of the position of the surgical device relative to the target anatomy so as to prevent the operator from feeling disoriented within the surrounding organs. Such disorientation could potentially cause unnoticed major organ damage. This guidance is particularly important during soft tissue surgery, which involves large-scale and rapid tissue deformations. A high update frequency above 1 kHz is required to maintain smooth and steady manipulation guidance. Due to its intrinsic complexity and this real-time requirement, PQ is computationally challenging. Various approaches have been proposed to achieve the required update rate [2], [7], with objects represented in specific formats such as spheres, torus or convex surfaces. The only attempts that apply PQ to haptic rendering, while considering explicitly the interaction of the body with the surrounding anatomical regions, involve modelling the anatomical pathway or the robotic device as a tubular structure [4], [8]. The computation burden is increased by the need to compute the placement of anatomical model relative to the robot whose shape is represented by more than 1 million vertices.

Due to its compute-intensive nature, PQ can greatly benefit from hardware acceleration. However, the massive amount of floating-point computations constitute a long data-path which is resource-demanding. Even if we could implement the data-path in an FPGA, the acceleration would be restricted by low parallelism and clock frequency. This challenge limits the implementation of PQ on an FPGA.

In this paper, we derive a PQ formulation which allows objects to be represented in complex geometry with vertices. To leverage the advantages of FPGAs, function transformation eliminates iterative trigonometric functions such that the algorithm can be fully-pipelined. We increase data-path parallelism by adopting a reduced precision data format which consumes fewer logic resources than high precision. To maintain the accuracy of results, potential incorrect outputs are re-computed in high precision. We design a novel memory architecture for buffering potential outputs and maintaining streaming data-flow. We further exploit the run-time reconfigurability of FPGA to optimise precision dynamically. To the best of our knowledge, our work is the first to apply reconfigurable technology to narrow-phase PQ computation.

The contributions of this paper are as follows:

- A PQ formulation for calculating the relative placement of objects modelled by vertices with complex morphology, which facilitates restructuring of trigonometric and search functions to be amenable to parallel implementation in hardware.
- Enhanced parallelism by treating input points as a novel data structure propagating through pipelines, together with FPGA-specific optimisations such as adapting PQ to reduced precision arithmetic, supporting multiple precisions in a novel memory architecture, and automating precision management with run-time reconfiguration.
- Implementation in a reconfigurable platform with four FPGAs which is shown to be 478 times faster than a single-core CPU, 58 times faster than a 12-core CPU system, 9 times faster than a GPU, and 3 times faster than a 4-FPGA system implemented in double precision.

II. Background

In this section, we first provide a brief introduction to PQ. Then we review the bit-width optimisation techniques that inspired our research.

Fig. 1 illustrates two objects acting as inputs to the proposed PQ. The object shown on the left is bounded by a series of contours (cf. Definition 1), each of which is outlined by a set of vertex points. This object can be either a luminal anatomy or a robotic endoscope/catheter. On the right, the mesh comprises vertex points which represent the morphological structure of either the robot or the target anatomy in complex shape. The proposed PQ actually computes how much the mesh deviates beyond the volumetric pathway bounded along the contours.

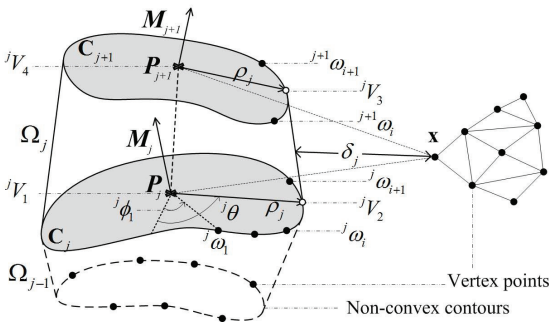
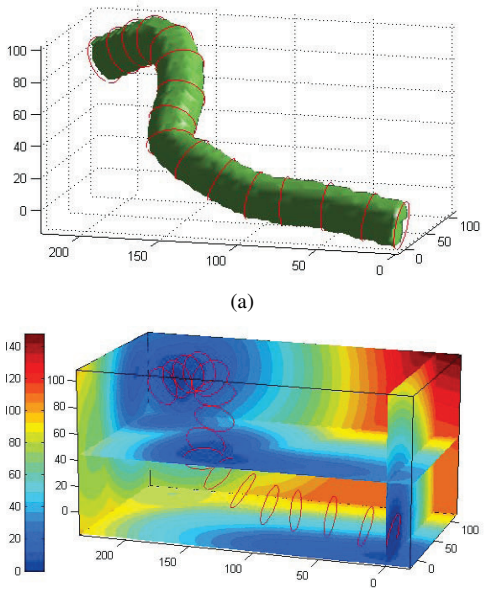


Fig. 1. (Left) Various sets of vertex points aligned on a series of contours; (Right) A set of vertex points located on an arbitrary form of mesh.

As shown in Fig. 2(a), a series of circular contours fitted along a part of an endoscope, which passes through the rectum up to the sigmoid colon. These contours form a constraint pathway. Fig. 2(b) shows a distance map in three-dimensional space with 177k grid points. Distance from every grid point to the endoscope is computed by the proposed PQ. The warmer colour, the further the point is located beyond the endoscope.

Definition 1. *Each contour is denoted by* C_j, $\forall j \in [1, ..., N_C]$. *A single segment* Ω_j *comprises two adjacent contours* C_j *and* C_{j+1}. P_j *is the centre of the contour* C_j. M_j *is the tangent of centre line of contour* C_j. ${}^j\omega_i = [{}^j\omega_{xi}, {}^j\omega_{yi}, {}^j\omega_{zi}]^T, (i = 1, ..., W)$ *are the contour vertices, where* W *is the number of vertex points outlining each contour.*



(a)

(b)

Fig. 2. (a) A virtual tube (in green) bounded by a series of contour (in red) denotes the configuration of an endoscope; (b) The corresponding three-dimensional distance map in grids of 86x48x43.

There has been previous work on hardware acceleration of board-phase PQ, which involves detecting collisions between primitive objects, e.g. spheres [7] or boxes [9]. Such an object can be a bounding volume tightly containing a union of multiple complex-shaped objects. On FPGA, the most relevant work is covered by Chow el at. [10]; however, to the narrow-phase PQ which computes the further detailed information, for instance, the shortest distance or penetration depth between polyhedra, GJK [11], V-Clip [12] and Lin-Canny [13] are the few well-established approaches, but their hardware acceleration is difficult due to algorithmic complexity. There is, thus far, no attempt of using FPGA. Such approaches are also restricted to the object represented in convex polyhedra. To this end, we have proposed a PQ approach for complex-morphology object [8] but how it can be incorporated with FPGA is not elaborated.

To leverage the advantages of FPGAs for hardware acceleration, Chow et al. [10] proposed a mixed precision methodology. They assume the data-path is short such that both the reduced precision and high precision implementations can be fitted in an FPGA. For complicated applications where the level of parallelism is limited by FPGA resource, their approach is not applicable. There are other studies about bit-width optimisation which uses minimum precision in a data-path given a required output accuracy. Examples include interval arithmetic [14], affine arithmetic [15], [16] and polynomial algebraic approach [17]. However, a reduction of precision in any stage within a data-path will result in a loss in output

accuracy which is uncorrectable. They require using accuracy models to relate output accuracy with the precisions of datapath. Our work is different from these work by deriving an automatic way to find an optimal precision using run-time reconfiguration.

III. Formulation of PQ

In this section, we derive our modified PQ process which was originally proposed in our previous work [8]. The significance of this modification is to formulate the PQ capable of processing the contours in complex shapes. As a result, it allows the analytical measure of the shortest Euclidean distance between an arbitrary set of vertices and a series of segments Ω_j (cf. Definition 1) which has been a well-known representation of a complex three-dimensional object [18]. Each segment is enclosed by two adjacent contours which are outlined by vertices arranged in polar coordinates; hence, it outperforms the existing narrow-phase PQs which are only compatible with convex objects.

In consideration of the point-to-segment distance, as shown in Fig. 1, four steps are taken to calculate the shortest distance δ_j between a point $\boldsymbol{x}$ and the corresponding edge ${}^jV_2 \rightarrow {}^j V_3$.

Before these steps, we capture the computation using polar coordinates. Given a contour C_j, ${}^j\phi_i$ are the polar angles corresponding to each contour vertex ${}^j\omega_i$. The polar angles of all the ${}^j\omega_i$ along the contour have to be computed. This computation can be further simplified by ignoring an axis coordinate. The poles and the contour vertices are then projected either on X-Y, Y-Z or X-Z plane based on the following conditions:

$$\begin{aligned}
&\text{if } |M_{zj}| = \max\left(|M_{xj}|, |M_{yj}|, |M_{zj}|\right)\\
&\quad {}^j\omega_i' = [{}^j\omega_{1i}, {}^j\omega_{2i}]^T = [{}^j\omega_{xi}, {}^j\omega_{yi}]^T, P_j' = [P_{xj}, P_{yj}]^T\\
&\text{if } |M_{xj}| = \max\left(|M_{xj}|, |M_{yj}|, |M_{zj}|\right)\\
&\quad {}^j\omega_i' = [{}^j\omega_{1i}, {}^j\omega_{2i}]^T = [{}^j\omega_{yi}, {}^j\omega_{zi}]^T, P_j' = [P_{yj}, P_{zj}]^T\\
&\text{if } |M_{yj}| = \max\left(|M_{xj}|, |M_{yj}|, |M_{zj}|\right)\\
&\quad {}^j\omega_i' = [{}^j\omega_{1i}, {}^j\omega_{2i}]^T = [{}^j\omega_{zi}, {}^j\omega_{xi}]^T, P_j' = [P_{zj}, P_{xj}]^T
\end{aligned} \tag{1}$$

Then ${}^j\phi_i$ is calculated as follows:

$$\overline{{}^j\omega_i'} = {}^j\omega_i' - P_j' \ , \qquad {}^j\phi_i = atan2\left(\overline{{}^j\omega_{2i}}, \overline{{}^j\omega_{1i}}\right) \tag{2}$$

We will explain the details of $atan2$ is Section IV-A.

Step 1: Find the normal of a plane containing points $\boldsymbol{x}$, P_j and P_{j+1}. The symbol $\times$ denotes a cross product of two vectors in three-dimensional space.

$$n_j = (P_j - x) \times (P_{j+1} - x) \tag{3}$$

Step 2: Calculate vectors ρ_j and ρ_{j+1} which are respectively perpendicular to tangents M_j and M_{j+1} and are both parallel to the plane with normal n_j.

$$\rho_j = n_j \times M_j \ , \qquad \rho_{j+1} = n_j \times M_{j+1} \tag{4}$$

Step 3: Determine a 4-vertex polygon outlined by ${}^jV_{i=1\ldots4} \in \Re^{3\times1}$ which is a part of the cross-section of segment Ω_j. This section is cut by a plane containing the point $\boldsymbol{x}$ and the line segment $P_j \rightarrow P_{j+1}$.

$$\begin{aligned}
{}^jV_1 &= P_j & {}^jV_2 &= P_j + t_j \cdot \rho_j\\
{}^jV_4 &= P_{j+1} & {}^jV_3 &= P_{j+1} + t_{j+1} \cdot \rho_{j+1}
\end{aligned} \tag{5}$$

At this stage, we need to calculate t_j and t_{j+1}. This can be achieved by mapping the values of ρ_j to a two-dimensional plane.

$$\begin{aligned}
&\text{if } |M_{zj}| = \max\left(|M_{xj}|, |M_{yj}|, |M_{zj}|\right)\\
&\quad \rho_j' = [\rho_{1j}, \rho_{2j}]^T = [\rho_{xj}, \rho_{yj}]^T\\
&\text{if } |M_{xj}| = \max\left(|M_{xj}|, |M_{yj}|, |M_{zj}|\right)\\
&\quad \rho_j' = [\rho_{1j}, \rho_{2j}]^T = [\rho_{yj}, \rho_{zj}]^T\\
&\text{if } |M_{yj}| = \max\left(|M_{xj}|, |M_{yj}|, |M_{zj}|\right)\\
&\quad \rho_j' = [\rho_{1j}, \rho_{2j}]^T = [\rho_{zj}, \rho_{xj}]^T
\end{aligned} \tag{6}$$

Then we calculate ${}^j\theta$, the corresponding polar angle of ρ_j' by Equation 7.

$${}^j\theta = atan2\left(\rho_{2j}, \rho_{1j}\right) \tag{7}$$

A search is performed to find ${}^j\phi_i$ and ${}^j\phi_{i+1}$ which embrace ${}^j\theta$. The polar angles ${}^j\phi_i$ and ${}^j\phi_{i+1}$ are calculated from Equation 2.

Based on the value i obtained from the search, t_j is calculated.

$$\begin{aligned}
a &= [(P_j - {}^j\omega_i)({}^j\omega_{i+1} - {}^j\omega_i)][({}^j\omega_{i+1} - {}^j\omega_i)\rho]\\
b &= [(P_j - {}^j\omega_i)\rho]\|{}^j\omega_{i+1} - {}^j\omega_i\|^2\\
c &= \|\rho\|^2\|{}^j\omega_{i+1} - {}^j\omega_i\|^2 - \|({}^j\omega_{i+1} - {}^j\omega_i)\rho\|^2\\
t_j &= \frac{a-b}{c}
\end{aligned} \tag{8}$$

Step 4: Define the shortest distance to be zero if the point $\boldsymbol{x}$ lies inside the polygon ${}^jV_{i=1\ldots4}$ on the same plane. Referring to [19], it can be determined by three variables $\lambda_{i=1,\ldots,3}$ calculated as follows:

$$\begin{aligned}
&\lambda_i = n_j \cdot \psi_i, i = 1, ..., 3\\
&\text{s.t. } \psi_i = ({}^jV_i - x) \times ({}^jV_{i+1} - x).
\end{aligned} \tag{9}$$

Here n_j denotes the normal defined in Equation 3 and ψ_i denotes the normal of the plane containing ${}^jV_{i=1\ldots4}$. For all $\lambda_{i=1,\ldots,3} \geq 0$, the shortest distance δ_j from point x to the segment Ω_j is assigned to zero such that $\delta_j(x) = 0$. Otherwise $\delta_j(x)$ will be considered as the distance from the point $\boldsymbol{x}$ to the line segment ${}^jV_2 \rightarrow {}^j V_3$. Such a point-line distance in three-dimensional space can be calculated simply referring to [20].

In consideration of many points and segments, Equation 10 generally expresses the deviation in distance from a single coordinate x_i to a series of constraint segments $(\Omega_1, ..., \Omega_{N_C-1})$, where $i = 1, ..., N_P$ and N_P is the total number of vertex points belong to the mesh model and $N_C - 1$ is the number of segments involved in the calculation.

$${}_i\delta_{N_C-1} = \min\left(\delta_1(\boldsymbol{x}_i), \delta_2(\boldsymbol{x}_i), ..., \delta_{N_C-1}(\boldsymbol{x}_i)\right) \tag{10}$$

The point with the maximum deviation, also known as penetration depth, is obtained below.

$$d^{N_C-1} = \max_{i=1,\ldots,N_P}\left({}_i\delta_{N_C-1}(\boldsymbol{x}_i)\right) \tag{11}$$

IV. Optimisation for Reconfigurable Hardware

The PQ formulation sketched in the previous section is not entirely hardware-friendly. In this section we discuss several techniques to allow PQ to benefit from FPGA technology.

A. Transformation of Trigonometric and Search Functions

The search process in step 3 of PQ checks whether ${}^j\phi_i \leq {}^j\theta$.

$$ {}^j\phi_i = atan2\left(\overline{{}^j\omega_{2i}}, \overline{{}^j\omega_{1i}}\right) \ , \ {}^j\theta = atan2\left(\rho_{2j}, \rho_{1j}\right) \tag{12} $$

$atan2(a,b)$ is not a hardware-friendly operator. It requires the calculation of $tan^{-1}(a,b)$ and then determines the appropriate quadrant of the computed angle based on the signs of a and b. $tan^{-1}(a,b)$ is expensive and is often not available in FPGA libraries, therefore, we transform Equation 12 to another form as shown below:

$$ \begin{aligned} {}^j\phi_i &= tan^{-1}\left(\frac{\overline{{}^j\omega_{2i}}}{\sqrt{\overline{{}^j\omega_{1i}}^2 + \overline{{}^j\omega_{2i}}^2} + \overline{{}^j\omega_{1i}}}\right) \\ {}^j\theta &= tan^{-1}\left(\frac{\rho_{2j}}{\sqrt{\rho_{1j}^2 + \rho_{2j}^2} + \rho_{1j}}\right) \end{aligned} \tag{13} $$

$atan2$ is transformed to tan^{-1} which is then cancelled out on both sides. As a result, the comparison becomes:

$$ \frac{\overline{{}^j\omega_{2i}}}{\sqrt{\overline{{}^j\omega_{1i}}^2 + \overline{{}^j\omega_{2i}}^2} + \overline{{}^j\omega_{1i}}} \leq \frac{\rho_{2j}}{\sqrt{\rho_{1j}^2 + \rho_{2j}^2} + \rho_{1j}} \tag{14} $$

In this case, square root calculation is much easier to be mapped to hardware.

B. Precision Optimisation

Reduced precision data-paths consume less logic resource at the expense of lower accuracy of results. To benefit from reduced precision data-paths without compromising accuracy, we partition the computation into two data-paths:

- Reduced precision data-path: Compute the deviations based on Equation 3 to 10.
- High precision data-path: Re-compute those deviations which are not accurate enough and calculate the penetration depth according to Equation 11.

In Equation 10, there are $\Delta m - 1$ comparisons involved to find the minimum value. The only item of interest is the minimum value ${}_i\delta_{\Delta m}$, rather than the exact values of every $\delta_j(\boldsymbol{x}_i)$. Based on this insight, we define the comparison operation:

$$ \begin{aligned} \delta^{min}_{1,\ldots,j} &= \min\left(\delta_1(\boldsymbol{x}_i), ..., \delta_j(\boldsymbol{x}_i)\right) \\ D &= \delta^{min}_{1,\ldots,j} - \delta_{j+1}(\boldsymbol{x}_i) \end{aligned} \tag{15} $$

The values of D when computed in reduced and high precision are denoted as D_{p_L} and D_{p_H}, respectively. D_{p_L} might have a flipped sign compared with D_{p_H}. We use the following three steps to make sure the results of Equation 10 is correct.

1) Evaluate Equation 15 using a reduced precision data format.
2) Estimate the maximum and minimum values of the value in high precision, i.e. $min(D_{p_H})$ and $max(D_{p_H})$, as shown in Equation 16.
   $E_{p_L}(\delta_{j+1}(\boldsymbol{x}_i))$ is the absolute error of $\delta_{j+1}(\boldsymbol{x}_i)$ in reduced precision p_L. It is computed at run-time and the details will be discussed later.
   $$ \begin{aligned} E_{p_L}(D_{p_L}) &= E_{p_L}(\delta^{min}_{1,\ldots,j}) + E_{p_L}(\delta_{j+1}(_i)) \\ \min\left(D_{p_H}\right) &= D_{p_L} - E_{p_L}(D_{p_L}) \\ \max\left(D_{p_H}\right) &= D_{p_L} + E_{p_L}(D_{p_L}) \end{aligned} \tag{16} $$
3) Determine whether the comparison result should be re-computed or dropped.
   Case A: $\min\left(D_{p_H}\right) > 0$, $\delta_{j+1}(\boldsymbol{x}_i)$ is smaller.
   Case B: $\max\left(D_{p_H}\right) < 0$, $\delta^{min}_{1,\ldots,j}$ is smaller.
   Case C: Cannot determine which value is smaller. Store both values for re-computation using high precision p_H.

In case A and B, the difference between the values is large enough to distinguish the sign of D_{p_H} even in the presence of errors introduced by reduced precision computations. In case C, the difference is small compared with the uncertainty introduced and therefore re-computation in high precision is necessary. The frequency of case C is lower than case A and B, therefore the performance gain from using reduced precision outweighs the re-computation overhead.

C. Dynamic Optimisation

We optimise the error bound based on feedback from run-time environment. Although the error bound $E_{p_L}(D_{p_L})$ can be derived statically [15], the estimated error bound grows pessimistically as it propagates along the data-path. Thus, we calculate the error bound using run-time data y and relative error RE_{p_L}. RE_{p_L} is profiled using a number of test vectors relative to a double precision data-path.

$$ E_{p_L}(y) = y \cdot RE_{p_L} \tag{17} $$

On the other hand, we need to decide the precision used in the reduced precision data-paths. A lower precision increases the level of parallelism and hence increases the throughput of reduce precision data-path. However, it increases the ratio of re-computation and the total run-time. It is important to find an optimal for the best performance. The ratio of re-computation is data-dependent which changes over time and cannot be computed in advance.

We propose a method to search for the optimal precision at run-time. When a new data set is applied or the ratio of re-computation exceeds a threshold, Algorithm 1 is invoked on the CPU to reconfigure the FPGA with a higher precision. The computation overhead of the algorithm is negligible. $T(p_L)$ is the run-time measured computation time when using precision p_L as the reduced precision.

V. Reconfigurable System Design

In this section, we present our design which treats input points as a data stream that propagates through the customised system architecture. We also propose an analytical model for performance estimation.

Algorithm 1 Run-time tuning of precision

1: Get the list of precisions P
2: $T(p_{test}) \leftarrow \infty$
3: **repeat**
4: $\quad T(p_L) \leftarrow T(p_{test})$
5: $\quad p_{test} \leftarrow \min(p \in P)$
6: $\quad$ Remove p_{test} from P
7: $\quad$ Configure the FPGA with precision p_{test}
8: $\quad$ Compute PQ and get $T(p_{test})$
9: **until** $T(p_{test}) > T(p_L)$

A. Streaming Data Structure

In PQ, there are N_P points to represent a mesh. PQ computes the shortest distance from each point to the segment boundary defined by N_C contours. An intuitive implementation is to stream one point into the FPGA at the beginning, then the contours are streamed in the subsequent N_C iterations. In other words, Equation 3 to 10 are iterated for $N_C - 1$ times. However, since every comparison operation in Equation 10 takes $L_{Cmp} > 1$ clock cycles to compute, the next comparison can only start after the current one completes. This significantly reduces the FPGA's throughput for L_{Cmp} times because the pipeline is not fully-filled.

To tackle this problem, we propose a data structure for efficient streaming. As shown in Fig. 3, data are streamed in an order as indicated by the arrows. In each iteration of N_S cycles, $N_S > L_{Cmp}$ points are processed together as a group. A new contour value is streamed in at the beginning of each iteration. In this manner, N_S points are being processed together in the pipeline to retain one output per clock cycle.

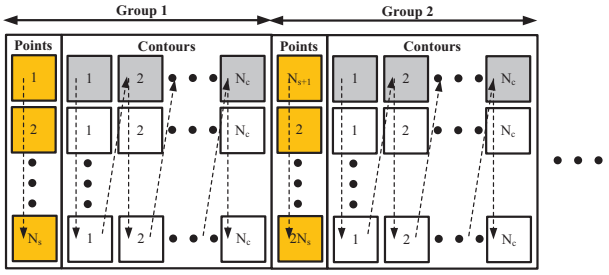


Fig. 3. Data structure: N_S points are processed in a group. Each point of a group is iterated for N_C times. Data are streamed in an order as indicated by the arrows.

B. System Architecture

Fig. 4 shows our proposed system architecture which consists of three major components.

Data-paths: As mentioned in Section IV, we employ reduced precision on FPGA to compute the deviations. The high precision data-path on CPU re-computes the deviations which are not sufficiently accurate, and then it calculates the penetration depth based on the minimum deviation. The reduced precision and high precision data-paths are interfaced by a comparator and a memory architecture as described below.

Comparator: The comparator compares the values of two deviations and determines which one is smaller. The FIFO stores the latest minimum deviation which corresponds to a group of points. The FIFO has N_S slots because N_S points are processed together. Since the deviations are calculated in reduced precision, according to Section IV-B, either one of the three conditions happens: (A) The distance from the data-path is smaller; (B) The distance stored in the FIFO is smaller; (C) The difference between the two distances is too small, so re-computation in high precision is necessary.

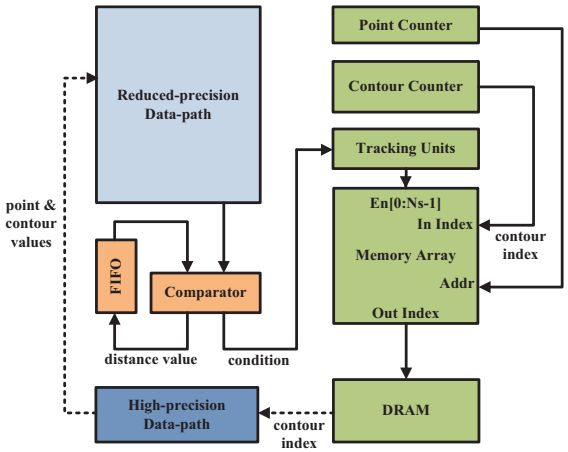
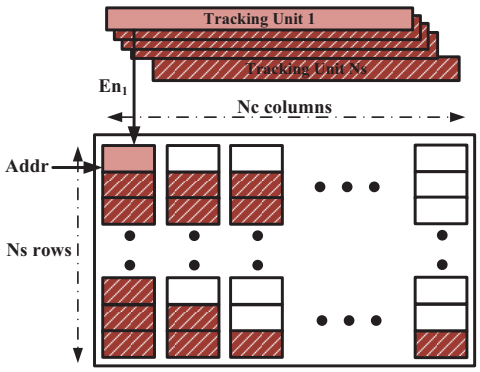


Fig. 4. System architecture: Solid lines represent communication on the FPGA board while dotted lines represent the bus connecting the reduced precision data-path on FPGA to the high precision data-path on CPU.

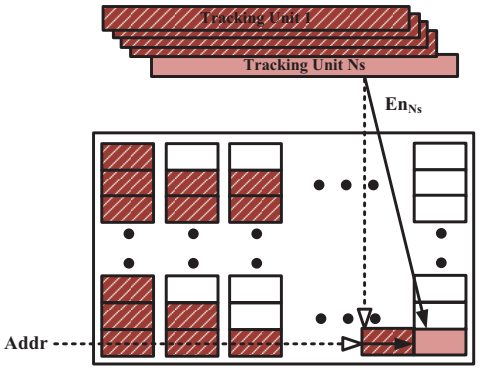
Memory Architecture: The purpose of the memory architecture is to store the contours that require re-computation. We design a memory array as shown in Fig. 5. There are N_S rows, each of which corresponds to the computation of one point which is addressed by a *point counter*. Each row consists of N_C elements and it serves as a buffer for contours that may need re-computation. Instead of storing the contours in three-dimensional coordinate, we store their indices to save memory space. The indices are counted by a *contour counter*. There are N_S *tracking units*, each for one row, to keep track of the latest elements where the indices should be written.

To understand the mechanism of memory architecture, consider the example in Fig. 5(a). First, the deviation in distance of *point 1* is being calculated. If the comparator indicates *condition A*, the value from the reduced precision data-path is the smallest, and all previous values stored in that row will be cleared. Second, the index corresponding to the new value is written to *element 1* of *row 1*. Third, *tracking unit 1* is updated to point to that element. If *condition B* is indicated, the minimum value is already stored in the memory and no update is required. Consider another example in Fig. 5(b) where the calculation of *point* N_S indicates *condition C*. Both the indices in the memory and from the data-path should be stored. Thus, a contour index is written to the next element and *tracking unit* N_S advances one element further.

After a group of points are processed, the contour indices stored in the memory array are transferred to the high precision data-path. To fully utilise the memory bandwidth, only non-empty memory columns are transferred in burst to the DRAM on the FPGA board.



(a) Condition A: the value from the reduced precision data-path is the smallest, *tracking unit 1* points to the *element 1* of *row 1*. Previous vales stored in *row 1* are cleared.



(b) Condition C: both the value in the memory and the index from the data-path should be stored. A contour index is written to the next element and *tracking unit* N_S advances one element further.

Fig. 5. Memory array stores contour indices for re-computation.

C. Performance Estimation

We derive a performance model to make the most effective use of the FPGA's resources. The results will be presented in Section VI-B and VI-C. The total computation time T_{Comp} is affected by the time spent on three parts: (1) the reduced precision data-path on FPGA, (2) the high precision data-path on CPU, (3) the data transfer through the bus connecting the CPU to FPGA. Equation 18 shows the three parts respectively.

$$T_{Comp} = T_{p_L} + T_{p_H} + T_{Tran} \tag{18}$$

As shown in Equation 19, the computation time of FPGA depends on the number of points N_P and the number of contours N_C. L_{p_L} is the length of the data-path but this term is usually negligible when compared with the amount of data being processed. Each point needs L_{Output} cycles to output indices on the memory array to DRAM. L_{Output} is affected by the bit-width available between the FPGA and the DRAM and their relations are shown in Equation 20.

$$T_{p_L} = \frac{N_P \cdot (N_C + L_{Output})}{freq_{p_L} \cdot N_{p_L}} + L_{p_L} \tag{19}$$

$$L_{Output} = \frac{N_C}{N_{Output}}, \quad N_{Output} = \frac{W_{DRAM}}{W_{Idx} \cdot N_{p_L}} \tag{20}$$

The computation time of CPU is related to the amount of data and the ratio of re-computation.

$$T_{p_H} = \alpha \cdot R \cdot N_P \cdot N_C \tag{21}$$

The data transfer time from the DRAM to CPU is judged by the amount of data, the ratio of re-computation and the bandwidth of the bus connecting the CPU to FPGA.

$$T_{Tran} = \frac{R \cdot N_P \cdot N_C \cdot W_{Idx}}{BW_{bus}} \tag{22}$$

TABLE I
PARAMETERS OF THE PERFORMANCE MODEL

N_P	Num. of points
N_C	Num. of contours
N_{p_L}	Num. of reduced precision data-path
N_{p_H}	Num. of high precision data-path
L_{p_L}	Length of the data-path
N_{Output}	Num. of outputs per data-path per cycle
L_{Output}	Num. of output cycles
R	Ratio of re-computation
W_{DRAM}	Bit-width of FPGA-DRAM connection
W_{Idx}	Bit-width of one contour index
$freq_{p_L}$	Clk. freq. of reduced precision data-path
α	Empirical constant of CPU speed
BW_{bus}	Bandwidth of the bus connecting the CPU to FPGA

VI. Experimental Evaluation

A. General Settings

We use the MPC-C500 reconfigurable system from Maxeler Technologies for our evaluation. The system has four MAX3 cards, each of which has a Virtex-6 XC6VSX475T FPGA with 476,100 logic cells and 2,016 DSPs. The cards are connected to two Intel Xeon X5650 CPUs and each card communicates with the CPUs via a PCI Express gen2 x8 link. The CPUs have 12 physical cores and are clocked at 2.66 GHz. We develop the FPGA kernels using MaxCompiler which adopts a streaming programming model and it supports customisable floating-point data formats.

We also build a CPU-based system by implementing the PQ formulation on a platform with two Intel Xeon X5650 CPUs running at 2.66 GHz. The code is written in C++ and compiled by Intel C compiler with the highest optimisation. OpenMP library is used to parallelise the program for multiple cores. IEEE double precision floating point numbers are used.

For the GPU-based system, we use an NVIDIA Tesla C2070 GPU which has 448 cores running at 1.15 GHz.

Our PQ implementation supports 100 contours and we set an update rate of 1 kHz as the real-time requirement.

B. Parallelism versus Precision

Fig. 6 shows the overall computation time (T_{Comp}) and the degree of parallelism of PQ versus different number of mantissa bits. Please note that all different configurations of mantissa bits have the same output accuracy. The data set includes 73k points and 100 contours. The computation times are obtained using our analytical model in Section V-C and they are verified experimentally using the implementation. The degree of parallelism is obtained by filling the FPGA with data-paths until the logic cell utilisation exceeds 80% after the

placement and routing process. The degree of parallelism is the highest when we start with four mantissa bits. Using more mantissa bits decreases the parallelism as well as the ratio of re-computation, therefore T_{p_L} increases but T_{p_H} decreases. As shown by the dotted line in the figure, a minimum computation time is achieved when 10 mantissa bits are used. Note that when the number of mantissa bits is more than 36, only one data-path can be mapped onto the FPGA. In such cases, we can implement the data-path in double precision directly which does not require any re-computation on CPU. This is indicated by the last data points of both curves.

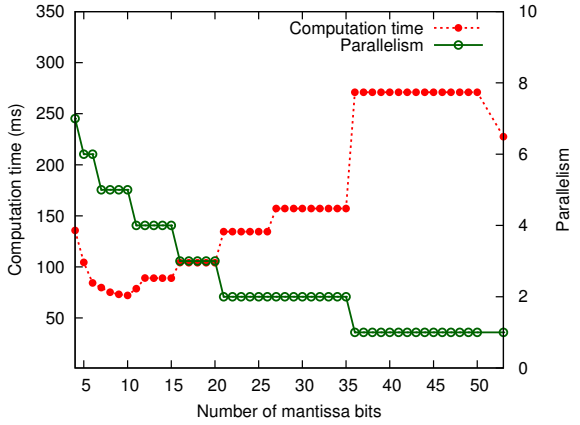


Fig. 6. Computation time [dotted line] and the level of parallelism [solid line] vs. different number of mantissa bits.

C. Ratio of Re-computation versus Precision

The dotted line in Fig. 7 shows the ratio of re-computation versus the number of mantissa bits. The results are obtained from a software version of PQ implementation with precisions adjusted using MPFR library [21]. For each point, 100 computations of deviation in distance are required. The ratio of re-computation drops exponentially as the number of mantissa bits increases. From the performance perspective, to the left the ratio of re-computation is too high, to the right the decrease of re-computation cannot offset the impact brought by the decrease in parallelism. When the number of mantissa bits is four, in average 2.66 out of 100 computations need to be re-computed using high precision, i.e. the ratio of re-computation is 2.66%. When the number of mantissa bits is greater then 15, the ratio of re-computation drops to 1% which is the minimum value as only one out of 100 values is re-computed. The last data points of both curves indicate the situation when double precision is used on the FPGA and no re-computation is necessary.

The solid line in Fig. 7 shows the number of point processed in 1 ms versus the number of mantissa bits. The application has a real-time update requirement of 1 kHz so the results are updated every 1 ms. The number of required vertex points is based on the user specification of the model resolution in three-dimensional space. When the number of mantissa bits is 10, the maximum number of points can be processed. It is because the throughput is the highest by balancing the ratio

TABLE II
COMPARISON OF PQ COMPUTATION IN 1 MS USING CPU-BASED SYSTEM (CPU), GPU-BASED SYSTEM (GPU), DOUBLE PRECISION FPGA-BASED SYSTEM (FPGA DP) AND FPGA+CPU SYSTEM WITH REDUCED PRECISION (FPGA RP)

	CPU	GPU	FPGA DP	FPGA RP
Clock freq. (MHz)	2,660	1,150	80	130 & 2,660 [a]
Num. of cores	12	448	4	20
Num. of mantissa bits	53	53	53	10 & 53 [b]
Num. of p_L eval. (k)	0	0	0	1009.4
Num. of p_H eval. (k)	173	106	320	10.1
Num. of total eval. (k)	173	106	320	1019.5
Eval. in p_H (%)	100	100	100	1
Num. of points in 1 ms	173	1,064	3,200	10,094
Normalised speedup	1x	6.15x	18.5x	58.35x
Reduced precision gain	-	-	1x	3.15x

[a] FPGA and CPU clock frequencies.
[b] Reduced precision and high precision.

of re-computation and the degree of parallelism. Since more points can be processed in real-time, we can handle a more complex robot model with a finer resolution.

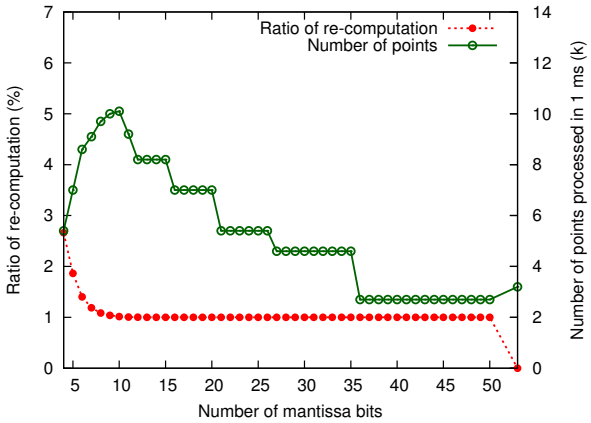


Fig. 7. Ratio of re-computation [dotted line] and the number of points processed in 1 ms [solid line] vs. different number of mantissa bits.

D. Comparison: CPU, GPU and FPGA

Table II compares the performance of PQ running on CPU, GPU and FPGA in double precision arithmetic, and our proposed reconfigurable system with CPUs and FPGAs.

In 1 ms, our proposed system is able to process 58 times more points than a 12-core CPU system, and 9 times more points than a GPU system. Without any optimisation, we can only implement one double precision data-path on an FPGA. Our proposed approach can support five reduced precision data-paths to be implemented in parallel on one chip, i.e. 20 data-paths in total on the 4-FPGA system. The clock frequency is also higher because reduced precision simplifies routing of signals. The performance gain over a double precision FPGA implementation is over 3 times.

Fig. 8 shows the computation time for a PQ update against the number of vertex points. The black solid line indicates the real-time bound of 1 ms. In the CPU-based system, even with the fastest configuration (12 cores), only 173 points can be processed in real-time. Meanwhile, the performance of our

proposed 1-FPGA system is on-par with a 4-FPGAs system in double precision. Our proposed 4-FPGAs system can process 10,094 points within the 1 ms interval.

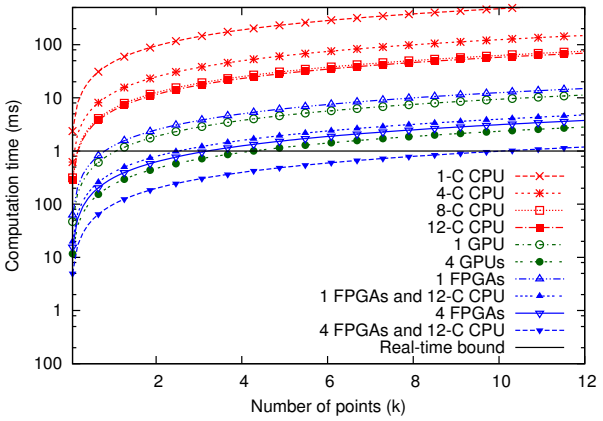


Fig. 8. Computation time for a PQ update with 100 contours vs. the number of points.

VII. Conclusion

This paper presents a reconfigurable computing solution to proximity query computation. To the best of our knowledge, our approach is the first to apply FPGAs to this problem.

We transform the algorithm to enable pipelining and apply reduced precision methodology to maximise parallelism. Run-time reconfiguration is employed to optimise precision automatically. We then map the optimised algorithm to a reconfigurable system with four Virtex-6 FPGAs and 12 CPU cores. Our proposed system achieves 478 times speedup over a single-core CPU, 58 times speedup over a 12-core CPU system, 9 times speedup over a GPU, and 3 times speedup over an FPGA implementation in double precision. Since more points can be processed in real-time, we can handle a more complex robot model with a finer resolution.

The work shows the potential of reconfigurable computing for PQ. Real-time performance is the pre-requisite to enable Dynamic Active Constraints, which has drawn increasing attention for effective human-robot collaborative control. Future work includes extending the current run-time reconfigurable architecture to cover other real-time applications that can benefit from the reduced precision approach. These applications, such as the real-time PQ between a robotic device and a rapidly deforming anatomy, will help us to evaluate the impact of run-time reconfiguration on various data sets. We are currently extending this work to cover imaged-guided catheterisation, particularly for cardiac electrophysiology intervention. To deal with the rapid deformation of the heart and the associated vessels, it is vitally important to provide the operator of a surgical robot online intra-operative guidance in real time, for which fast and efficient PQ computation is essential.

Acknowledgment. This work is supported in part by the European Union Seventh Framework Programme under grant agreement number 248976, 257906 and 287804, by UK EPSRC, by Maxeler University Programme, by Xilinx, and by the Croucher Foundation.

References

[1] E. Gilbert and C.-P. Foo, "Computing the distance between general convex objects in three-dimensional space," *IEEE Trans. Robotics and Automation*, vol. 6, no. 1, pp. 53–61, 1990.

[2] N. Chakraborty, J. Peng, S. Akella, and J. Mitchell, "Proximity queries between convex objects: An interior point approach for implicit surfaces," *IEEE Trans. Robotics*, vol. 24, no. 1, pp. 211–220, 2008.

[3] K.-W. Kwok, V. Vitiello, and G.-Z. Yang, "Control of articulated snake robot under dynamic active constraints," in *Proc. Int. Conf. Medical image computing and computer-assisted intervention*, 2010, pp. 229–236.

[4] M. Li, M. Ishii, and R. Taylor, "Spatial motion constraints using virtual fixtures generated by anatomy," *IEEE Trans. Robotics*, vol. 23, no. 1, pp. 4–19, 2007.

[5] D. Constantinescu, S. Salcudean, and E. Croft, "Haptic rendering of rigid contacts using impulsive and penalty forces," *IEEE Trans. Robotics*, vol. 21, no. 3, pp. 309–323, 2005.

[6] M. Jakopec, F. Rodriguez y Baena, S. Harris, P. Gomes, J. Cobb, and B. L. Davies, "The hands-on orthopaedic robot "acrobot": Early clinical trials of total knee replacement surgery," *IEEE Trans. Robotics and Automation*, vol. 19, no. 5, pp. 902–911, 2003.

[7] M. Benallegue, A. Escande, S. Miossec, and A. Kheddar, "Fast c1 proximity queries using support mapping of sphere-torus-patches bounding volumes," in *Proc. Int. Conf. Robotics and Automation*, 2009, pp. 483–488.

[8] K.-W. Kwok, K. H. Tsoi, V. Vitiello, J. Clark, G. C. T. Chow, W. Luk, and G.-Z. Yang, "Dimensionality reduction in controlling articulated snake robot for endoscopy under dynamic active constraints," *IEEE Trans. Robotics*, vol. 29, no. 1, pp. 15–31, 2013.

[9] X. Zhang and Y. Kim, "Interactive collision detection for deformable models using streaming aabbs," *IEEE Trans. Visualization and Computer Graphics*, vol. 13, no. 2, pp. 318–329, 2007.

[10] G. C. T. Chow, K. W. Kwok, W. Luk, and P. H. W. Leong, "Mixed precision processing in reconfigurable systems," in *Proc. Int. Symp. Field-Programmable Custom Computing Machines*, 2011, pp. 17–24.

[11] E. Gilbert, D. Johnson, and S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.

[12] B. Mirtich and B. Mirtich, "V-clip: Fast and robust polyhedral collision detection," *ACM Trans. Graphics*, vol. 17, pp. 177–208, 1998.

[13] M. Lin and J. Canny, "A fast algorithm for incremental distance calculation," in *Proc. Int. Conf. Robotics and Automation*, 1991, pp. 1008–1014.

[14] C. F. Fang, R. A. Rutenbar, and T. Chen, "Fast, accurate static analysis for fixed-point finite-precision effects in dsp designs," in *Proc. Int. Conf. Computer-aided Design*, 2003, pp. 275–282.

[15] D.-U. Lee, A. Abdul Gaffar, W. Luk, and O. Mencer, "MiniBit: Bit-width optimization via affine arithmetic," in *Proc. Design Automation Conf.*, 2005, pp. 837–840.

[16] W. G. Osborne, J. Coutinho, R. C. C. Cheung, W. Luk, and O. Mencer, "Instrumented multi-stage word-length optimization," in *Proc. Int. Conf. Field-Programmable Technology*, 2007, pp. 89–96.

[17] D. Boland and G. Constantinides, "Automated precision analysis: A polynomial algebraic approach," in *Proc. Int. Symp. Field-Programmable Custom Computing Machines*, 2010, pp. 157–164.

[18] J. Ponce, D. Chelberg, and W. Mann, "Invariant properties of straight homogeneous generalized cylinders and their contours," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 11, no. 9, pp. 951–966, 1989.

[19] F. Preparate and M. Shamos, *Computational Geometry*. Springer, 1985.

[20] Point-line distance–3-dimensional. MathWorld. [Online]. Available: http://mathworld.wolfram.com/Point-LineDistance3-Dimensional.html

[21] L. Fousse, G. Hanrot, V. Lefèvre, P. Péissier, and P. Zimmermann, "MPFR: A multiple-precision binary floating-point library with correct rounding," *ACM Trans. Math. Softw*, vol. 33, no. 2, pp. 13:1–13:15, 2007.