Parallel Motion Planning Using Poisson-Disk Sampling

Chonhyon Park, *Student Member, IEEE*, Jia Pan, *Member, IEEE*, and Dinesh Manocha, *Fellow, IEEE*

***Abstract*—We present a rapidly exploring-random-tree-based parallel motion planning algorithm that uses the maximal Poisson-disk sampling scheme. Our approach exploits the free-disk property of the maximal Poisson-disk samples to generate nodes and perform tree expansion. Furthermore, we use an adaptive scheme to generate more samples in challenging regions of the configuration space. The Poisson-disk sampling results in improved parallel performance and we highlight the performance benefits on multicore central processing units as well as manycore graphics processing units on different benchmarks.**

***Index Terms*—Motion planning, parallel algorithm, Poisson-disk sampling.**

I. Introduction

SAMPLING-BASED approaches are widely used to compute collision-free paths for motion planning. The most influential sampling-based motion planning schemes include probabilistic roadmaps (PRM) [1] and rapidly exploring random trees (RRT) [2]. The key idea in these planners is to generate samples in the free configuration space of the robot and connect them with collision-free edges to construct a graph. PRM planners are mostly used for multiple-query planners and involve considerable preprocessing in terms of roadmap computation. On the other hand, most motion planning applications do not perform multiple queries. These situations arise when the robot does not know the entire environment *a priori*, or when it moves to a new environment. In such cases, incremental sampling-based algorithms, such as RRT, are widely used. The RRT algorithm has been extended in several aspects for use in systems with differential constraints, nonlinear dynamics, and hybrid systems. Moreover, it has also been integrated with physical robot platforms.

The simplest RRT algorithms are based on generating uniform random samples and connecting the nearby samples until a collision-free path from the initial configuration to the goal configuration has been computed. In this paper, we present a novel approach that uses Poisson-disk samples for RRT planners and constructs the trees using parallel algorithm.

Poisson-disk sampling is a well-known scheme that can be used in high dimensions to generate a random set of points with two properties: the points are tightly packed together, yet remain separated from each other by a specified minimum distance [3]–[5]. Poisson-disk distributions are known to have good blue-noise characteristics and are widely used in statistics, computer graphics, mesh algorithms, artificial intelligence, image processing, and random object placement. Poisson-disk sampling is a sequential random process for selecting points in a region. The sampling process is maximal if no more points can be added, which implies that the entire region is completely covered by the disks of radius r centered at each sample.

In this paper, we present an extended version of our Poisson-RRT algorithm that was originally introduced in a conference paper [6]. The algorithm uses precomputed maximal Poisson-disk samples to generate an RRT tree. We use an adaptive sampling scheme that increases the sampling rate in the challenging regions of the configuration space (e.g., narrow passages).

Main results: The main contribution of the paper is that the use of one-time precomputation of the maximal Poisson-disk samples result in 1) fewer redundant nodes in the configuration space and 2) are more amenable to parallelization on commodity multicore central processing units (CPUs) and manycore graphics processing units (GPUs). We highlight the performance of our multithreaded CPU- and GPU-based implementations on well-known benchmarks and demonstrate improved parallelization as compared to prior parallel RRT algorithms.

The rest of this paper is organized as follows. In Section II, we survey prior work on RRT-based motion planning and highlight properties of Poisson-disk sampling. We give an overview of our Poisson-RRT planning algorithm in Section III. We present the details of the planning algorithm in Section IV and describe the parallel extension in Section V. Theoretical analysis of the algorithm is given in Section VI. We highlight the performance on different benchmarks of low- and high-dimensional spaces in Section VII.

II. Related Work and Background

In this section, we give a brief overview of prior work on RRT-based motion planning, parallel planning algorithms, Poisson-disk sampling, and Lattice-based Sampling.

Manuscript received January 6, 2016; revised May 28, 2016; accepted October 13, 2016. Date of publication December 19, 2016; date of current version April 3, 2017. This paper was recommended for publication by Associate Editor K. E. Bekris and Editor C. Torras upon evaluation of the reviewers' comments. This work was supported by the Army Research Office under Contract W911NF-10-1-0506; by the National Science Foundation under Award 1000579, Award 1117127, and Award 1305286; and by Sandia Laboratories. A preliminary version of this paper was presented in part at the IEEE International Conference on Robotics and Automation, Hong Kong, China, May 2014.

C. Park and D. Manocha are with the Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC 27599 USA (e-mail: chpark@cs.unc.edu; dm@cs.unc.edu).

J. Pan is with the Department of Mechanical Engineering and Biomedical Engineering, City University of Hong Kong, Hong Kong (e-mail: jiapan@cityu.edu.hk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TRO.2016.2632160

A. RRT-Based Motion Planning

There is extensive work on RRT-based motion planning due to its efficiency. The original RRT algorithm [7] grows the RRT tree based on the Voronoi property, biasing the search toward unexplored regions of free configuration space. Many variants to improve this original RRT have been proposed. Dynamic-domain RRT [8] adaptively controls the Voronoi bias of the nodes, which results in a better exploration. Some algorithms [9], [10] use workspace information to guide the growth of the RRT tree. RESAMPL [11] adaptively chooses different sampling strategies for RRT according to the local properties of different regions. Shkolnik and Tedrake's Ball Tree algorithm [12] adaptively approximates the free configuration space using hyperspheres with varying radii. Visibility of the expanded nodes is used to adjust the tree expansion in adaptive RRT [13]. RRT has also been extended for optimal or near-optimal motion planning [14]–[16].

B. Parallel Planning Algorithms

The performance of motion planning can be improved using the parallel computation. There are many planning approaches that exploit distributed clusters or shared-memory systems or commodity parallel processors.

1) Algorithms for Distributed Clusters: Distributed clusters have been widely used for solving compute-intensive problems. Clusters are defined as a large number of connected machines or nodes, where each of them has local memory, and a big computational problem is divided into small pieces and assigned to different processors in the cluster for parallel computation.

Many parallel techniques have been proposed to improve the performance of planning using distributed clusters. Lozano-Pérez and O'Donnell [17] compute the primitive map of a 3-D configuration space using parallel computation. Amato and Dale [18] propose a parallel PRM planning approach, which has scalable speedups.

Many planning algorithms exploit parallelism based on subdividing the configuration space [19] and use clusters to expand the tree in a different region of the configuration space. Different subdivision techniques have been proposed for roadmap-based planning [20] or tree-based planning algorithms [21], [22].

Some approaches combine PRM and RRT in order to use the massive parallelism [23]. Many parallel RRT algorithms can be classified into AND parallelization and OR parallelization [24]. AND parallelization uses multiple cores to expand a single tree by adding multiple new nodes in parallel. The multinode expansion of AND parallelization allows the tree to expand faster. However, it generates redundant nodes, which can causes the performance degradation due to the collision checking and nearest neighbor search overheads. In OR parallelization, each thread running on a separate core maintains its own tree and solves the motion planning problem independently. OR parallelization improves the average cost of the solution [25], [26], but it usually expects less speedup than AND parallelization

2) Algorithms for the Shared-Memory System: Nowadays, commodity processors in a single machine have multiple cores. Although these systems have fewer cores and overall processing power as compared to large distributed clusters, multiple threads running on such shared-memory processors have access to the same memory, and there is no major overhead of transferring the data between the nodes in a cluster. It is especially useful for parallel algorithms of RRT, which does not have the massive parallelism like the graph construction step of PRM. Many AND and OR parallel RRT algorithms are proposed for shared-memory systems [27], [28]. AND parallel approaches on shared-memory systems have better efficiency than clusters because the multiple threads can share the same tree data structure on shared memory [29]. Updates of the shared tree require synchronization, and the performance can be improved using lock-free data structures [30].

Many approaches also exploit manycore GPUs for accelerating the planning algorithms. Pisula *et al.* [31] use the rasterization hardware for improving the sample generation in narrow passages. Recently, the general-purpose GPU technology allows efficient use of the GPUs using appropriate interfaces (e.g., CUDA, OpenCL). G-Planner [32] uses manycore GPU processors to parallelize and accelerate PRM approach. Kider *et al.* [33] propose a GPU-based R* algorithm for 6-degree-of-freedom (DOF) problems. Park *et al.* [34] use multiple CPU and GPU cores for parallel optimization-based planning. Bialkowski *et al.* [35] use multiple cores on GPUs to perform parallel collision checking along different edges of RRT.

3) Our Approach: Our planning algorithm is based on AND parallelization on the shared-memory system. However, our approach has a better performance than the original AND parallel RRT by guiding the tree expansion using Poisson-disk samples to reduce the generation of redundant nodes.

C. Maximal Poisson-Disk Sampling

Poisson-disk sampling [3]–[5] ensures that each sample is at least a minimum distance, r, from the other samples. Each sample has an associated disk, which is a hypersphere of radius r, and no additional samples can be placed in the disk. The area of a disk (or the volume of the hypersphere) is called the coverage volume of the associated sample. Maximal Poisson-disk sampling requires that there is no room or space to place a new Poisson-disk sample in the domain, i.e., the entire domain is covered by the disks of samples. Fig. 1(a) shows a set of maximal Poisson-disk samples for the same domain. Overall, maximal Poisson-disk sampling satisfies following properties in any dimensions:

$$\begin{aligned} &\text{free-disk:} && \forall \mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}, \mathbf{x}_i \neq \mathbf{x}_j : \|\mathbf{x}_i - \mathbf{x}_j\| \geq r \\ &\text{maximal:} && \forall \mathbf{x} \in \Omega, \exists \mathbf{x}_i \in \mathbf{X} : \|\mathbf{x} - \mathbf{x}_i\| < r \end{aligned} \tag{1}$$

where $\mathbf{X} = \{\mathbf{x}_i\}$ is the set of samples in domain Ω. Given a nonmaximal sampling, a new Poisson-disk sample can be generated in a bias-free manner, i.e., the probability of selecting a sample from any uncovered subregion is proportional to the subregion's volume:

$$\forall A \in S(\mathbf{X}) : \mathbb{P}(\mathbf{x} \in A) = \frac{|A|}{|S(X)|} \tag{2}$$

where $S(\mathbf{X}) = \{\mathbf{x} \in \Omega : \|\mathbf{x} - \mathbf{x}_i\| \geq r, \forall \mathbf{x}_i \in \mathbf{X}\}$ is the region uncovered by existing disks. For one Poisson-disk sample $\mathbf{x}$, another Poisson-disk sample $\mathbf{y}$ is its neighbor if the disks

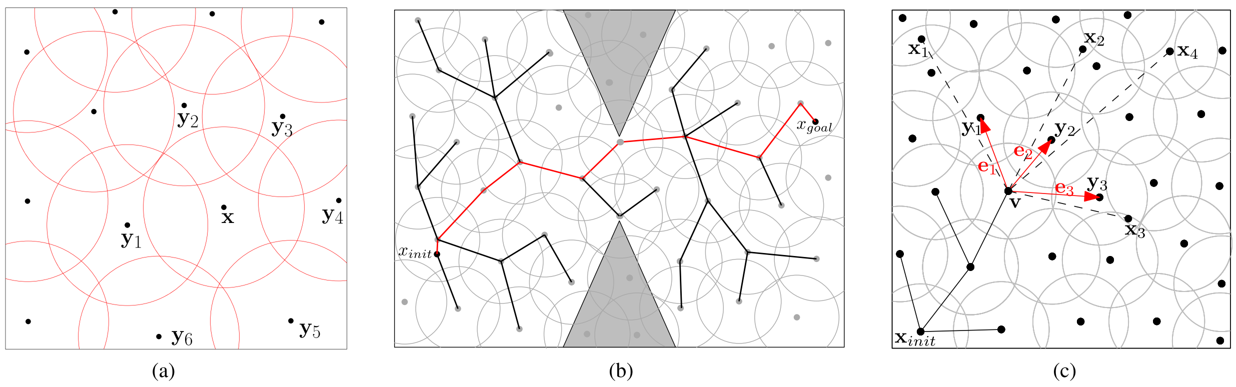


Fig. 1. (a) Maximal Poisson-disk sampling. Each black point is a Poisson-disk sample and the red circle is the corresponding Poisson disk. $\mathbf{y}_i$ are the neighbors of $\mathbf{x}$. (b) Poisson-disk sampling is used to generate the RRT tree and compute a collision-free path from $\mathbf{x}_{\text{init}}$ to $\mathbf{x}_{\text{goal}}$. (c) Parallel Poisson-RRT tree expansion using four threads. The ith thread expands the tree toward sample $\mathbf{x}_i$, $i = 1, 2, 3, 4$. The red vectors $\mathbf{e}_i$ show the new RRT edges added. Since $\mathbf{x}_2$ and $\mathbf{x}_4$ correspond to the identical Poisson-disk sample ($\mathbf{y}_2$), both of them result in adding the edge $\mathbf{e}_2$ to the tree. There is no redundant node added to the tree.

corresponding to the two samples overlap, i.e., $\|\mathbf{x} - \mathbf{y}\| < 2r$, as shown in Fig. 1(a).

These properties are useful when maximal Poisson-disk samples are used for the RRT algorithm. The free-disk property ensures that the new sample is not too close to an existing node in the RRT tree, which thereby ensures good coverage of the free space. For a fixed number of samples, the maximal property generates the best distribution of samples in the configuration space. Furthermore, we use an adaptive scheme based on Poisson-disk samples that makes it possible to find paths in challenging areas or in narrow passages of the configuration space, as described in Section IV-C. The bias-free property of the Poisson-disk samples (which functions similarly to the Voronoi diagram bias used in the original RRT algorithm [7]).

However, the computation of the Poisson-disk samples that satisfy the maximal property is compute intensive. Most of the sampling algorithms have been practical only for 2-D or 3-D spaces [36], or up to 6-D space [37]. Recently, relaxed sampling algorithms are suggested for high-dimensional space [38].

Our planning algorithm does not depend on a particular Poisson-disk sampling algorithm because it uses samples that are precomputed in offline. We use exact [37] and relaxed [38] maximal Poisson-disk sampling approach for our 6-D and 23-D benchmarks in Section VII, respectively.

D. *Lattice-Based Sampling*

Although random sampling is widely used in motion planning, many other sampling techniques have been proposed [39]. Grid-based sampling is used in many applications due to its low dispersion, which implies that the samples are generated in such a manner that the largest uncovered area in the configuration space is as small as possible, and that the size of the uncovered space is governed by the grid resolution. However, grid-based approaches generate samples that are aligned with the coordinate axis; these aligned samples are undesirable, as they increase the variance in the planning algorithm's running time [40]. Lattices are a generalization of grids that allow nonorthogonal axes or other spatial decompositions; common lattices include the Sukharev grid and the nongrid lattice, both of which give samples with low dispersion, low discrepancy, and low environmental sensitivity. Discrepancy is a criterion that measures the largest axis-aligned rectangular area, which is not covered by samples. Multiresolution approaches [41], [42] are used to increase the number of samples in lattice-based planning algorithms and have been combined with replanning [43]. As compared to grid-based samples, Maximal Poisson-disk samples have low dispersion and low discrepancy, and in addition, the resulting samples are not aligned with any axes.

III. Overview

Our goal is to use Poisson-disk sampling as the underlying sample generation process for RRT-based planning. The nodes of the RRT tree correspond to Poisson-disk samples, and the tree expansion step can be performed in parallel using multiple threads. In this section, we give an overview of the proposed algorithm.

A. *Assumptions and Notations*

The configuration $\mathbf{x}$ of a robot is a point in a configuration space $\mathcal{C}$, which consists of collision-free region $\mathcal{C}_{\text{free}}$ and $\mathcal{C}$-obstacle region $\mathcal{C}_{\text{obs}}$; our goal is to find a continuous collision-free path from an initial configuration $\mathbf{x}_{\text{init}}$ to a goal configuration, $\mathbf{x}_{\text{goal}}$.

The RRT tree $\mathbf{T}$ is initialized with the root node of $\mathbf{x}_{\text{init}}$, and the algorithm expands the tree incrementally. Each iteration of RRT planning executes two main procedures.

1) *Sampling*: The `sample()` procedure generates a new random configuration $\mathbf{x}$, which determines the direction of the tree expansion.
2) *Expansion*: The `expansion()` procedure includes two steps: 1) nearest node search and 2) local planning. Given a configuration $\mathbf{x}$, nearest node search finds a node $\mathbf{v}$ in $\mathbf{T}$: the closest node to $\mathbf{x}$ according to the given metric of the configuration space, ρ (e.g., the weighted Euclidean metric). For high-dimensional space, approximate algorithms [44] with computational complexity $\mathcal{O}(d \log n)$ are used, where d is the dimension of the configuration space.

```
Input: start configuration $\mathbf{x}_{init}$, goal configuration $\mathbf{x}_{goal}$,
    precomputed Poisson-disk (radius $r$) sample set $\mathbf{X}$
Output: RRT Tree $\mathbf{T}$
1: $\mathbf{T}$.add($\mathbf{x}_{init}$)
2: $\mathbf{X}$.add($\mathbf{x}_{init}$)
3: /* Can handle multiple threads easily */
4: for $i = 1$ to $m$ do in parallel
5:     while $\mathbf{x}_{goal} \notin \mathbf{T}$ do
6:         $\mathbf{x} \leftarrow$ sample()
7:         $\mathbf{T} \leftarrow$ extendMPDS($\mathbf{T}, \mathbf{x}, \mathbf{X}$)
8:     end while
9: end for
```

Fig. 2. RRT planning using maximal Poisson-disk sampling.

The local planning step checks whether the shortest path between $\mathbf{v}$ and $\mathbf{x}$ lies in $\mathcal{C}_{\text{free}}$ (i.e., that the configuration of the path does not collide with the obstacles). If the path is collision-free, $\mathbf{x}$ is added to $\mathbf{T}$ as a new node connected to the node $\mathbf{v}$. If the path has a collision, the collision-free configuration $\mathbf{x}_{\text{new}}$ on the path that is farthest from $\mathbf{v}$ is added to $\mathbf{T}$ instead of $\mathbf{x}$.

B. RRT Planning Using Maximal Poisson-Disk Sampling

The RRT algorithm is efficient for single-query problems, since the algorithm incrementally expands the RRT tree to the unexplored regions and terminates when the solution is found. However, this incremental expansion of the tree means that it is difficult to make an efficient parallel algorithm for planning. The AND parallelization can expand the tree faster than the original RRT. However, as the number of threads increases, the algorithm results in more redundant nodes in the RRT tree, degenerating the performance of overall planning.

The overall Poisson-RRT algorithm is shown in Fig. 2. In order to lessen the overhead caused by the redundant nodes, our algorithm uses precomputed Poisson-disk samples in the tree expansion. The precomputed samples satisfy the free-disk property in (2), where $\mathbf{X}$ is set of samples and r is a predefined minimum distance between any of two samples. Our algorithm maintains the RRT Tree data structure $\mathbf{T}$ and the Poisson-disk sample set $\mathbf{X}$ during the planning, and each node $\mathbf{v}$ in $\mathbf{T}$ has a pointer to the corresponding Poisson-disk sample $\mathbf{x}$ in $\mathbf{X}$ to access additional information (e.g., neighboring samples or the disk radius) needed in the `extendMPDS()` procedure.

Unlike the standard RRT, which performs local planning between the nearest node $\mathbf{v}$ and the configuration $\mathbf{x}$, `extendMPDS()` procedure in Fig. 4 chooses a Poisson-disk sample $\mathbf{x}_{\text{nbr}}$ that is closest to $\mathbf{x}$ among $\mathbf{v}$'s neighboring Poisson-disk samples. The free-disk property ensures that the chosen sample is at least a minimum distance, denoted here by r, from $\mathbf{v}$. If the local planning finds a collision-free path between $\mathbf{x}_{\text{nbr}}$ and $\mathbf{v}$, $\mathbf{x}_{\text{nbr}}$ is added to the RRT tree as a new node. The tree expansion is repeated until the goal configuration $\mathbf{x}_{\text{goal}}$ is added to the tree. Our approach eliminates the problem of multiple threads of the algorithm choosing the same direction, which generates redundant nodes that are too close to each other in the standard RRT tree expansion. In our algorithm, the threads that choose the same direction do not generate redundant nodes; instead, they choose the same Poisson-disk sample and stop the redundancy problem from developing. We add the sample only once to the tree. An example of tree construction in our algorithm is shown in Fig. 1(c).

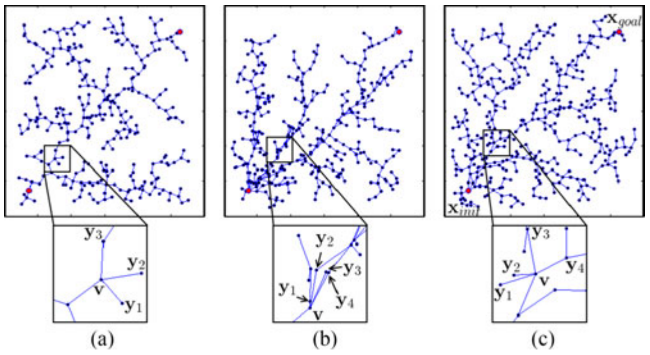


Fig. 3. Comparison of RRT trees generated using different planning approaches. (a) Tree corresponding to the original RRT algorithm is generated according to the Voronoi bias of the sequential algorithm. (b) Parallel RRT tree generated by AND parallelism has many redundant nodes that are close to other nodes in the tree (e.g., the new nodes $\mathbf{y}_2$, $\mathbf{y}_3$, and $\mathbf{y}_4$ are close to $\mathbf{y}$). (c) Tree generated with Poisson-disk sampling has fewer redundant nodes due to the free-disk property of samples, although it is generated using the parallel sampling.

Fig. 3 shows the RRT trees generated by an original RRT, an AND parallelization RRT, and our algorithm. The tree generated by AND parallelization has many redundant nodes that are close to other tree nodes, while the tree generated using Poisson-disk sampling has efficiently spaced nodes.

IV. Poisson-RRT Algorithm

In this section, we present the details of our planning algorithm, including precomputation of maximal Poisson-disk samples, tree expansion, and adaptive sampling.

A. Precomputation of Maximal Poisson-Disk Samples

As a precomputation step, Poisson-disk samples are generated in the d-dimensional configuration space. The computation of Poisson-disk samples is time consuming; however, these samples are independent of obstacles and do not need to be recomputed frequently. We can use a precomputed sample set computed offline for multiple planning queries and only need to recompute the precomputed samples when the number of DOFs of the robot is changed, which would rarely occur.

Although our planning algorithm is not dependent on a specific Poisson-disk sampling algorithm, the experimental results of our paper is based on the parallel version of the sampling algorithm proposed by Ebeida *et al.* [37]. For a given disk radius r, Ebeida *et al.*'s algorithm generates uniform base grids that cover the entire configuration space $\mathcal{C}$. Each grid cell is a square with the side length $r/\sqrt{d}$, and each cell can contain at most one sample. The algorithm repeatedly subdivides grid cells, which are not fully covered by Poisson-disks, and generate samples in those cells.

The complexity of the algorithm is linear in the number of generated samples, which is exponential to the number of dimensions and the disk radius. The parallel version of the algorithm improves the sampling performance by processing multiple grid cells simultaneously.

B. Tree Expansion

Given a new random sample $\mathbf{x}$, our algorithm extends the planning tree $\mathbf{T}$ using the `extendMPDS()` procedure, which is summarized in Fig. 4.

For a sample point $\mathbf{x}$, the algorithm finds the nearest tree node $\mathbf{v}$ in $\mathbf{T}$. From a node $\mathbf{v}$, the Poisson-RRT algorithm chooses a sample $\mathbf{x}_{\mathrm{nbr}}$, which is a point closest to $\mathbf{x}$ among $\mathbf{v}$'s neighboring Poisson-disk samples. These steps utilize the nearest neighbor search.

It is possible that a Poisson-disk sample can be chosen by more than one thread in the nearest neighbor search (lines 1and 2). However, when the algorithm adds samples (line 6), it prevents adding a sample in $\mathbf{X}$ to $\mathbf{T}$ more than once. This approach helps the algorithm to avoid adding redundant nodes while using the parallel tree extension.

The `collisionCheck()` procedure, used for local planning, checks for collisions along the edge that joins $\mathbf{v}$ to $\mathbf{x}_{\mathrm{nbr}}$. Only the precomputed Poisson-disk samples can be added as a node to $\mathbf{T}$. However, the precomputed Poisson-disk samples may not have a large-enough number of samples to find a collision-free solution. It requires a way to find a path that does not passes the precomputed samples if a collision is detected. As a result, the algorithm performs adaptive Poisson subsampling at runtime to generate more samples with reduced distance between them (lines 8–17).

C. Adaptive Sampling

As shown in lines 8–17 in Fig. 4, we perform adaptive sampling in the regions where the local planning algorithm finds a collision between an edge of the tree and an obstacle; in that subregion of the configuration space, we generate additional samples with a reduced disk radii. This process is illustrated in Fig. 5. In the initialization of the data structure $\mathbf{X}$, all samples $\mathbf{x}$ have the same disk radius r. If the `collisionCheck()` procedure detects a collision during local planning along the edge that joins $\mathbf{v}$ to $\mathbf{x}_{\mathrm{nbr}}$, the procedure computes $\mathbf{x}_{\mathrm{free}}$ as the last collision-free point in the direction from $\mathbf{v}$ to $\mathbf{x}_{\mathrm{nbr}}$. If a collision occurs within the disk associated with $\mathbf{v}$ or $\mathbf{x}_{\mathrm{nbr}}$, the adaptive sampling algorithm reduces the radius of the disk by half (lines 9–16 in Fig. 4). This reduction changes some regions that were covered in the original disk to become uncovered [see Fig. 5(b)]. Therefore, we need to generate samples to cover these regions.

The computation of maximal Poisson-disk samples can be slow and is performed only during the preprocessing, and not at runtime. Instead of using the exact Poisson-disk sampling algorithm, we generate new samples by precomputed templates. In the precomputation step, we compute n templates from the Poisson-disk sample set, which is also used for adaptive sampling. In each template, there is a sample with radius $r/2$ placed at the origin. We use the same algorithms [37], [38], used for the

```
Input: RRT Tree $\mathbf{T}$, a new random sample $\mathbf{x}$, Poisson-disk sample set $\mathbf{X}$
Output: RRT Tree $\mathbf{T}$
1: $\mathbf{v} \leftarrow$ nearestNode($\mathbf{T}, \mathbf{x}$)
2: $\mathbf{x}_{nbr} \leftarrow \mathrm{argmin}_{\mathbf{y} \in \mathbf{v}\text{'s neighbor}} \rho(\mathbf{y}, \mathbf{x})$
3: (success, $\mathbf{x}_{free}$) $\leftarrow$ collisionCheck($\mathbf{v}, \mathbf{x}_{nbr}$)
4: if success then
5:     /* no collision along that edge */
6:     $\mathbf{T}$.add($\mathbf{x}_{nbr}$)
7: else
8:     /* if there is collision, perform adaptive sampling */
9:     if $\rho(\mathbf{x}_{free}, \mathbf{v}) < \mathbf{v}.r$ then
10:        /* If the collision occurs in the disk of $\mathbf{v}$, reduce the coverage of $\mathbf{v}$
11:        $\mathbf{v}.r \leftarrow \mathbf{v}.r/2$
12:    end if
13:    /* If the collision occurs in the disk of $\mathbf{x}_{nbr}$, reduce the coverage of $\mathbf{x}_{nbr}$ */
14:    if $\rho(\mathbf{x}_{free}, \mathbf{x}_{nbr}) < \mathbf{x}_{nbr}.r$ then
15:        $\mathbf{x}_{nbr}.r \leftarrow \mathbf{x}_{nbr}.r/2$
16:    end if
17:    $\mathbf{X}$.add(adaptiveSampling($\mathbf{v}, \mathbf{x}_{nbr}, \mathbf{x}_{free}$))
18: end if
```

Fig. 4. RRT tree `extendMPDS()` procedure using maximal Poisson-disk sampling.

sample precomputation, to add maximal Poisson-disk samples of radius $r/2$ within a hypersphere of radius r and dimension d.

In a runtime adaptive sampling (line 17 in Fig. 4) for $\mathbf{v}$, we randomly select one of n templates and scale it to make the samples in the template have the same radius of $\mathbf{v}$. We rotate the scaled template for a randomly selected orientation [45] and add samples in the rotated template to $\mathbf{X}$, except the sample at the origin which has the same position with $\mathbf{v}$ [see Fig. 5(c)]. Using the positions of $\mathbf{v}$ and $\mathbf{x}_{\mathrm{free}}$, we compute which sample in the template is closest to $\mathbf{x}_{\mathrm{free}}$ when the template is applied to $\mathbf{v}$. The new sample is connected to $\mathbf{T}$ for future expansion if there is no collision on the local path joining $\mathbf{v}$ and the new sample [see Fig. 5(d)].

This adaptive sampling approach locally breaks the free-disk property, but allows the algorithm to handle any width of narrow passages, since it adaptively generates more samples in the difficult regions of the configuration space.

V. Parallelization of the Algorithm

In sampling-based planning algorithms like RRT, the most of the computation time is spent in the nearest neighbor search or collision checking procedures, as shown in Fig. 9. This section presents how our parallel planning algorithm accelerates those procedures in two ways: 1) we use AND parallel RRT which provides high parallelism to utilize the computational resources, and 2) we use parallel GPU algorithms for nearest neighbor search and collision checking.

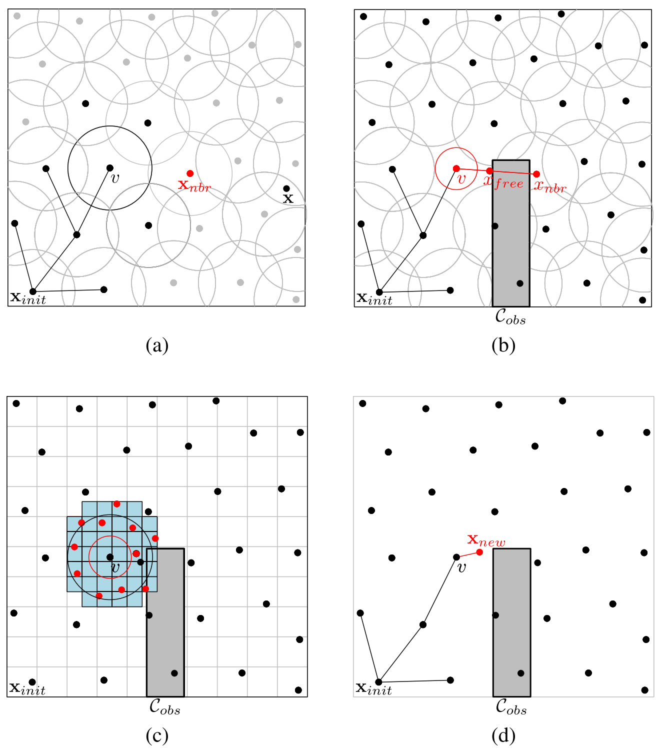


Fig. 5. Tree extension and adaptive sampling. (a) Sample $\mathbf{x}_{\text{nbr}}$ is the point closest to $\mathbf{x}$ among the neighboring Poisson-disk samples of $\mathbf{v}$. (b) If $\overline{\mathbf{v}\mathbf{x}_{\text{nbr}}}$ intersects $\mathcal{C}_{\text{obs}}$, collisionCheck() procedure returns the last collision-free point $\mathbf{x}_{\text{free}}$. If the collision occurs within the disk associated with $\mathbf{v}$, the radius of this disk is reduced by half. (c) Precomputed template of Poisson-disk samples is applied to $\mathbf{v}$ to find a point which is close to $\mathbf{x}_{\text{free}}$ and satisfies the maximal property in the disk of $\mathbf{v}$. (d) New sample $\mathbf{x}_{\text{new}}$ is connected to $\mathbf{T}$ if there is no collision on the local path joining $\mathbf{v}$ and $\mathbf{x}_{\text{new}}$.

A. Massively Parallel Computation

Our parallel planning algorithm can be implemented using either multicore CPUs or manycore GPUs to improve the overall performance, as described in Section VII. We assume a shared memory system, which is common for GPU-based algorithms and does not require message passing interfaces of distributed systems. Our algorithm achieves the massive parallelism by performing parallel `extendMPDS()` evaluations on multiple threads, as shown in Fig. 2. Theoretical analysis of the planning complexity improvement is given in Section VI.

However, parallel tree expansion results in synchronization issues among multiple threads, which shares the data structure $\mathbf{T}$ and $\mathbf{X}$. Furthermore, the Poisson-RRT algorithm needs to check whether the new node being added to the tree is already in the tree or not. Our data structures are designed to minimize the use of functions that can be used for thread synchronization, as they have additional overhead. A sample once added to $\mathbf{X}$ from either the initialization step or adaptive sampling is never modified or deleted, which requires the synchronization for accessing operations. One exception is that each sample has a marker that indicates whether the sample is already added as a tree node to $\mathbf{T}$ or not, which is used to prevent adding the sample multiple times to $\mathbf{T}$. A tree node $\mathbf{v}$ is added to $\mathbf{T}$ only if the marker of the corresponding sample $\mathbf{x}$ is `false`, and set it to `true`. However, it can be handled by an atomic compare-and-swap operation that is available on current commodity CPU and GPU processors, instead of locks.

B. Parallel Nearest Neighbor Search

As described in Section IV-B, the planning algorithm finds the nearest tree node for a sample by utilizing the nearest neighbor search, which has a computational complexity $\mathcal{O}(d \log n)$, where d is the dimension of the configuration space. There has been extensive work on nearest neighbor search using GPUs [32], [46], [47]. We use the algorithm proposed by Pan *et al.* [47], which uses locality-sensitive hashing (LSH) for clustering nearby points in high-dimensional spaces. The algorithm generates the same hash value for points near one another; points with the same hash value are stored in the same bucket of the hash table. Using this data structure, the nearest neighbor search for a point can be computed in nearly constant time since it requires only looking up one bucket in the hash table.

C. Parallel Collision Checking

In order to accelerate collision checking, we compute bounding volume hierarchies (BVH) for the robot and the obstacles in the environment. We construct the oriented bounding box (OBB) trees [48] for the triangle model representations of the robot and obstacles using a GPU-based construction algorithm [49]. The OBB trees improve the performance of collision checking because of their high culling efficiency.

When the tree node and the nearest Poisson-disk sample are computed, the algorithm performs local planning to check for a feasible path between the two configurations. We use discrete collision detection, which discretizes the path between two configurations into multiple steps, between the robot and obstacles; we then check for collisions during each step. GPU uses multiple threads to perform this multiple-step collision checking in parallel.

VI. Theoretical Analysis

In this section, we analyze the computational complexity of our parallel Poisson-RRT algorithm and compare it with the AND parallel RRT algorithm.

In order to compute the time complexity for RRT algorithms, we use the concept of attraction sequence borrowed from [2]. An attraction sequence is a finite sequence $\mathcal{A} = \{A_0, A_1, \ldots, A_k\}$ of sets with the following properties.

1) $A_0 = \{\mathbf{x}_i\}$ and $A_k = \{\mathbf{x}_{\text{goal}}\}$.
2) For each set A_i, there exists a set B_i, called the basin, such that for any $\mathbf{x} \in A_{i-1}$, $\mathbf{y} \in A_i$, and $\mathbf{z} \in \mathcal{C} \backslash B_i$, there is $\rho(\mathbf{x}, \mathbf{y}) < \rho(\mathbf{y}, \mathbf{z})$, where ρ is a metric defined in the configuration space.
3) For all $x \in B_i$, there exists an l such that the sequence of actions $\{u_1, \ldots, u_l\}$ selected by the RRT's `extend` algorithm or Poisson-RRT's `extendMPDS` algorithm will bring the state into $A_i \subseteq B_i$.

Intuitively, property 2 ensures that an element in B_i will always be selected by the nearest neighbor query `nearestN-`

`ode` in Fig. 4, and property 3 implies that B_i is a potential well and can attract the nearby states into A_i. Given a scenario, we should choose an attraction sequence with each node A_i as large as possible and the sequence length k as small as possible. The values of A_i's size and k provide a rough estimation about how difficult a scenario is for the motion planning. If the space is open, k would be small and each A_i would be large. If the space contains narrow passages, then k will be large and each A_i would be small. Since the values of A_i and k are only related to the scenarios and are independent with the underlying planning algorithms, they provide a consistent manner to compare different planning algorithms. In addition, the potential well property of B_i helps us to be free from the details of local planning or adaptive sampling in different approaches.

Given an attraction sequence A, let p be defined as $p = \min_i\{\mu(A_i)/\mu(\mathcal{C}_{\text{free}})\}$, which corresponds to a lower bound on the probability that a random state will lie in a particular region A_i. Here $\mu(A_i)$ represents the area of A_i. Based on the attraction sequence, we can compute the expected computational complexity for sequential RRT algorithms.

Theorem 6.1: If an attraction sequence of length k exists, then the expected time complexity of the sequential original RRT and Poisson-RRT planner is $\mathcal{O}(d\frac{k}{p})$.

Proof: The time complexity of sequential RRT planner includes three parts: random sample $\mathcal{O}(n)$, local planning $\mathcal{O}(c \cdot n)$, and nearest neighbor computation $\sum_{j=1}^{n} \mathcal{O}(d \log j) = \mathcal{O}(d \log(n!))$, where n is the number of iterations, c is the number of collision checking for a local planning, which is bounded as a constant with the maximum edge length $2r$, and d is the dimension of the configuration space. As a result, the overall time complexity is $\mathcal{O}(c \cdot n)$ if local planning dominates the overall complexity or $\mathcal{O}(d\log(n!))$ if nearest neighbor computation dominates the overall computation. Given an attraction sequence of length k, the probability that the planner can find a path after n iterations is $\binom{n-1}{k-1}p^k(1-p)^{n-k}$. If the complexity is dominated by nearest neighbor computation, the expected computational complexity of sequential RRT is

$$\begin{aligned}
T_{\text{RRT}} &= \sum_{n=k}^{\infty}\binom{n-1}{k-1}p^k(1-p)^{n-k}\mathcal{O}(d\log(n!)) \\
&\simeq \sum_{n=k}^{\infty}\binom{n-1}{k-1}p^k(1-p)^{n-k}\mathcal{O}(dn\log n) \\
&= \frac{dp^k k}{(1-p)^k}\mathcal{O}\left(\sum_{n=k}^{\infty}\binom{n}{k}(1-p)^n\log n\right).
\end{aligned}$$

Let $F(k) = \sum_n \binom{n}{k}(1-p)^n \log n$, then $F(k) \leq (1-p)\frac{\log(k+1)}{\log k}(F(k-1) + F(k))$, which implies that $\sum_n \binom{n}{k}(1-p)^n \log n \leq \frac{(1-p)^k}{(\frac{\log k}{\log(k+1)}-1+p)^{k+1}}$. As a result, the expected computational complexity is

$$T_{\text{RRT}} = \mathcal{O}\left(d\frac{k}{p}\frac{1}{(1-\frac{1}{p}(\frac{\log(k+1)}{\log k}-1))^{k+1}}\right) \simeq \mathcal{O}\left(d\frac{k}{p}\right).$$

If the timing cost of each iteration is dominated by local planning, it is easy to prove that the complexity is $T_{\text{RRT}} = \mathcal{O}(\frac{k}{p})$. ■

Next, we analyze the complexity of parallel RRT algorithms, which use m threads simultaneously for tree expansion. We first show that parallel Poisson-RRT algorithm can reduce the number of redundant tree nodes as compared with AND parallel RRT:

Theorem 6.2: If a parallel Poisson-RRT algorithm extends its tree by adding m nodes in parallel, then during each iteration, the expected number of tree nodes generated is $m' = \frac{1}{q}(1 - (1-q)^m)$, where $q = 1/N$ and N is the size of the Poisson-disk sample set $\mathbf{X}$.

Proof: When parallel Poisson-RRT extends the RRT tree, m different random samples are generated. Some of the samples may belong to the same Poisson-disks, while others may not. We now compute a bound on the expected number of distinct disks associated with these m samples.

Let Y be the number of distinct disks. Let D_i be 1 if one of the random samples is located inside the ith disk, and 0 if not. We have $\mathbb{E}(D_i) = 1 - \mathbb{P}(D_i = 0) = 1 - (1-q)^m$. Therefore, $\mathbb{E}(Y) = \mathbb{E}(\sum_{i=1}^{N} D_i) = N\mathbb{E}(D_i) = \frac{1}{q}(1-(1-q)^m)$.

According to the Poisson-RRT algorithm, a new node is generated for each distinct disk. Therefore, $m' = \mathbb{E}(Y)$. ■

Remark 6.3: For AND parallel RRT, m different tree nodes are generated during each iteration. Obviously, $m' \leq m$, so Poisson-RRT reduces the size of RRT tree. Moreover, Poisson-RRT can adaptively change the number of new tree nodes depending on whether the region is open or not. For a challenging region, Poisson-RRT uses a smaller r, which results in large q, and in this case, $m' \simeq m$, i.e., Poisson-RRT is similar to the AND parallel RRT. For an open region, Poisson-RRT will use a large r, which results in a small q and $m' \ll m$, i.e., parallel Poisson-RRT will generate a tree with fewer nodes.

The computational complexity of AND parallel RRT can be computed as follows.

Theorem 6.4: If an AND parallel RRT algorithm expands its tree by adding m nodes in parallel, the planning complexity is $\mathcal{O}((d + \log m)\frac{k}{1-(1-p)^m})$ if an attraction sequence of length k exists.

Proof: We define p' as the lower bound on the probability that one of the m random states generated during one iteration will lie in a particular region A_i. Then, $p' = 1 - (1-p)^m$. In a manner similar to that laid out in Theorem 6.2, if the timing cost is dominated by nearest neighbor computation, the expected time complexity is

$$\begin{aligned}
T_{\text{AND}} &= \sum_{n=k}^{\infty}\binom{n-1}{k-1}p'^k(1-p')^{n-k}\mathcal{O}(d\log(n!m^n)) \\
&\simeq \sum_{n=k}^{\infty}\binom{n-1}{k-1}p'^k(1-p')^{n-k}\mathcal{O}(n\log n + n\log m) \\
&= \mathcal{O}((d+\log m)\frac{k}{p'}).
\end{aligned}$$

When the timing cost is dominated by collision detection and local planning, the resulting bound is $\mathcal{O}(\frac{k}{p'})$. ■

Corollary 6.5: The planning complexity of the parallel Poisson-RRT algorithm is smaller than the complexity of the AND parallel RRT.

Proof: From Theorem 6.2, the expected number of tree nodes generated in an iteration is m' for the parallel Poisson-RRT algorithm. The complexity of parallel Poisson-RRT can be computed by substituting m in Theorem 6.4 with m'

$$\begin{aligned} T_{\text{Poisson}} &= \mathcal{O}\left((d + \log m')\frac{k}{p'}\right) \\ &\leq \mathcal{O}((d + \log m)\frac{k}{p'} = T_{\text{AND}}. \end{aligned} \tag{3}$$

■

Remark 6.6: When p is nearly 1, which corresponds to a relatively easy planning problem, then $T_{\text{Poisson}} \geq T_{\text{RRT}}$ and the speedup is small. When p is small, which corresponds to more challenging planning scenario, then $T_{\text{Poisson}} \simeq (d + \log m')\frac{k}{\text{mp}} = \frac{1}{m} T_{\text{RRT}}$.

The proofs in this section assume that the effect of the runtime adaptive sampling is neglectable in the planning performance. If the planner needs to generate a lot of samples using adaptive sampling, it would degrade the performance. However, in Section VII, we show in Table III that the adaptive sampling is used rarely and only a small number of samples are generated at runtime using adaptive sampling in our experiments. As a result, the analysis described in this section is applicable to most scenarios.

VII. Results

In this section, we present our experimental results and highlight the performance of our planning algorithm on different benchmarks. We implement the algorithm using Open Motion Planning Library (OMPL) [50]. For parallel implementations, we use Boost and NVIDIA CUDA libraries for CPU-based and GPU-based planners, respectively. All the timings described in this section were generated on a commodity PC with an Intel i7-2600 eight-core CPU and a NVIDIA GTX 680 GPU (for GPU-based Poisson-RRT).

A. *Poisson-RRT on OMPL Benchmarks*

For the first experiment, we used four well-known benchmark scenarios from OMPL, shown in Fig. 6. These planning problems are all in 3-D space, but vary in their complexities. Some have narrow passages and are more challenging than others.

For each benchmark, we evaluate the performance of our different GPU-based planner implementations, the GPU-based AND parallel RRT, and the parallel Poisson-RRT with the adaptive sampling. We set these GPU-based planners to expand 32 nodes in parallel, which can exploit GPU many-cores for the nearest neighbor search and the collision checking computations using parallel algorithms of LSH and OBB tree BVH. We compare the GPU-based planners with the following existing CPU-based RRT variant algorithms available in OMPL, and the details of the comparison are given in [51].

1) *Standard RRT (RRT-Extend) [2]:* Sequential RRT that uses random uniform sampling.

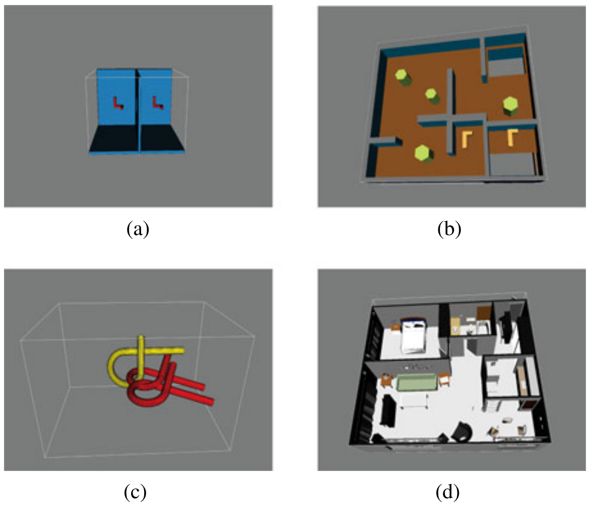


Fig. 6. Planning problems used as the benchmarks of various planners. Easy moves a robot from the left room to the right room by passing a window; Cubicles moves the robot in an office environment; Alpha puzzle contains a very narrow passage; Apartment moves the piano to the hallway near the door entrance. Alpha puzzle and Apartment benchmarks are relatively more challenging than Easy and Cubicles benchmarks. (a) Easy. (b) Cubicle. (c) Alpha Puzzle. (d) Apartment.

2) *RRT-Connect [7]:* Bidirectional algorithm that expands trees from both the initial and the goal configurations.
3) *Lazy-RRT:* A variant of RRT algorithm that defers collision checks until it finds a solution, which is based on Lazy-PRM [52] technique.
4) *pRRT [24]:* AND parallel RRT algorithm on CPU 8-cores.

The performance of RRT-based planning algorithms is governed by the maximum extension distance ϵ. A smaller ϵ needs to generate more nodes to find the solution, while a larger ϵ causes more failures in the local planning. Similarly, the performance of the Poisson-RRT algorithm is affected by the radius of the precomputed Poisson-disk samples r. We set the ϵ for different benchmarks using the default OMPL computation, which is proportional to the workspace size of the benchmark. We set $r = \frac{2}{3}\epsilon$ for Poisson-RRT algorithms.

The mean and standard deviation of the total time taken by the planner are shown in Table I. The means and standard deviations are computed from 100 trials for each benchmark. Fig. 7 shows the parallel algorithm's planning-time speedup on the OPML benchmarks as compared to the original CPU-based RRT algorithm, which uses a single core.

Based on these experimental results, we observe the following.

1) The performance of single-threaded Poisson-RRT is not always better than the original RRT, which is due to the required additional computations such as adaptive sampling.
2) In general, our GPU-based Poisson-RRT is faster than the original CPU-based algorithms, providing up to 25X speedup over the CPU algorithms.
3) The performance improvement of the parallel planners (pRRT and Poisson-RRT) over the sequential planners is more significant in narrow passage scenarios (AlphaPuzzle and Apartment) than in open-space scenarios (Easy

TABLE I
PERFORMANCE OF RRT-BASED PLANNING ALGORITHMS ON DIFFERENT BENCHMARKS

	CPU-based										GPU-based (32 threads)				
# of threads	single-threaded								8 threads		32 threads			
Algorithm	RRT		RRT-Connect		LazyRRT		Poisson-RRT		pRRT		pRRT		Poisson-RRT	
Benchmark	Mean	Std.dev.	Mean	Std.dev.	Mean	Std.dev.	Mean	Std.dev.	Mean	Std.dev.	Mean	Std.dev.	Mean	Std.dev.
Easy	0.34	(0.33)	0.12	(0.14)	0.12	(0.09)	0.37	(0.48)	0.18	(0.15)	0.04	(0.04)	0.03	(0.03)
Cubicle	2.31	(0.84)	0.53	(0.09)	81.54	(43.07)	4.03	(1.49)	0.59	(0.31)	0.63	(0.35)	0.31	(0.36)
AlphaPuzzle	32.76	(13.54)	19.92	(14.73)	72.72	(71.74)	27.23	(27.83)	6.69	(5.28)	1.93	(1.22))	1.31	(1.28)
Apartment	232.24*	(89.42)	20.15	(20.74)	11.55	(12.18)	72.54	(62.01)	126.68	(69.94)	19.97	(7.33)	11.88	(7.95)

We report planning time for each case. The mean and standard deviation are computed from 100 trials on each benchmark. CPU-based pRRT utilizes eight threads to fully exploit the eight-core CPU. GPU-based algorithms use 32 threads for the computation. *RRT algorithm cannot find solution in some instances and those are taken in account in computing the average.

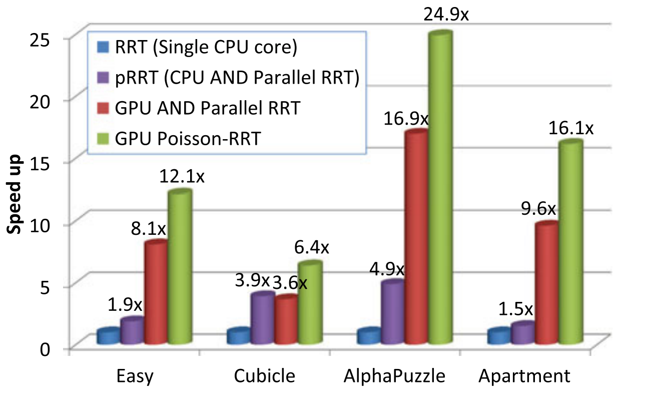


Fig. 7. Speedup of GPU-based algorithms from the original RRT algorithm, which uses a single CPU core. GPU-based Poisson-RRT improves the performance of CPU-based algorithm up to 25 times. Poisson-RRT provides additional 50–100% speedup as compared to the GPU-based AND Parallel RRT implementation. The CPU-based pRRT algorithm would expand eight nodes in parallel. The GPU-based implementation would expand 32 nodes in parallel, and also used GPU-based parallel nearest neighbor and collision detection algorithms.

and Cubicle). This is because the main advantage of parallel RRTs is their capability to perform exploration and exploitation simultaneously. Suppose there are two ways of connecting the initial and the goal configurations. One is closer to the initial configuration but has a narrow passage, and the other is further away but is more open. Sequential RRTs may get stuck before the narrow passage for a while and cannot make progress until they eventually find the open corridor. The parallel planners can perform exploration more efficiently and thus would find the further away open passage earlier than traditional RRTs. However, in open-space scenarios, the exploration advantage of parallel RRTs is less significant because sequential RRTs can easily find a solution without too much exploration, and the computational overhead of parallel RRTs becomes more important.

B. Comparison of Sampling Algorithms

In order to evaluate the benefit that comes solely from the use of Poisson-disk samples, we compare the performance of planners with different sampling algorithms. In addition to the

TABLE II
PERFORMANCE OF GPU RRT PLANNING WITH DIFFERENT SAMPLING ALGORITHMS

Sampling Algorithm	Poisson-disk Samples	Random Samples	Grid-based Samples	Hammersley Samples [53]
Benchmark	Mean (Std.dev.)	Mean (Std.dev.)	Mean (Std.dev.)	Mean (Std.dev.)
Easy	0.03 (0.03)	0.04 (0.04)	0.03 (0.02)	0.03 (0.03)
Cubicle	0.31 (0.36)	0.63 (0.35)	0.29 (0.10)	0.50 (0.31)
AlphaPuzzle	1.31 (1.28)	1.93 (1.22)	2.27 (0.54)	1.72 (0.74)
Apartment	11.88 (7.95)	19.97 (7.33)	16.04 (4.79)	15.66 (4.35)

Planners use the same nearest neighbor and collision checking algorithms.

Poisson-disk sampling and random sampling, we also evaluate the performance of grid-based and Hammersley [53] samples and modify Poisson-RRT to use grid-based or Hammersley samples instead of the precomputed Poisson-disk samples. For the grid-based samples, we generate precomputed samples on an axis-aligned grid with a cell size $2r/\sqrt{d}$ and generate samples in half-size grids when the adaptive sampling is required. For the Hammersley samples, which have an deterministic order satisfying the low discrepancy, we generate the same number of samples with the Poisson-RRT for the samples generated in the precomputation and the runtime. Table II shows the result of the four GPU-based planners, GPU Poisson-RRT, GPU AND Parallel RRT, and planners use grid-based and Hammersley samples. It shows that the use of Poisson-disk sampling improves 50–100% performance than the use of random samples, even though they use the same GPU parallelism. As mentioned in Remark 6.3 (see Section VI), the speedup is more significant in benchmarks with large open spaces. However, the performance of a planner that uses grid-based samples varies in different benchmarks. In particular, in the benchmarks where the axis-aligned grid-based samples can find a good solution, the performance of that planner with grid-based samples is close to that of the Poisson-RRT algorithm. However, the performance of the grid-based path planner is worse than random sampling in the Alpha puzzle benchmark. These benchmarks have a narrow passage that does not along with the axes, and therefore, grid-based planners do not perform well. The planner with Hammersley samples shows better performance in narrow passage scenarios, but not as good as the planner with Poisson-disk samples.

TABLE III
PERFORMANCE OF POISSON-RRT ALGORITHM WITH DIFFERENT SAMPLE RADII FOR "EASY" BENCHMARK [SEE FIG. 6(A)]

	Precomputed Sample Radius	Precomputed Samples	Precomputation Time (s)	Run-time Samples	Planning Time (s)
Adaptive Sampling	256	7.821	0.003	48.201	0.079
	128	40.780	0.008	17.636	0.029
	64	264.016	0.091	60.371	0.150
	32	2383.558	1.280	407.659	0.430
	16	16534.969	17.818	393.186	0.546
Uniform Sampling	128	40.780	0.008	0	22.284
	64	264.016	0.091	0	1.724
	32	2383.558	1.280	0	0.436
	16	16534.969	17.818	0	1.340

We compare the planning time of our adaptive sampling approach with a planner that only uses precomputed samples. We observe improved performance with our adaptive sampling approach.

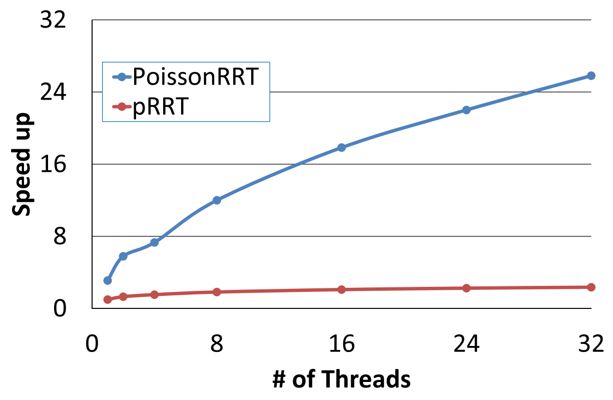


Fig. 8. Speedup of parallel Poisson-RRT with the number of parallel threads. The speedup is computed based on the sequential RRT algorithm for apartment benchmark.

In the next set of experiments, we compared the planning performance of precomputed Poisson-disk samples using different radii. We also compare our adaptive-sampling planners' planning time to that of the samplers using only the precomputed Poisson-disk samples. The result for benchmark "Easy" [see Fig. 6(a)] is shown in Table III. The uniform sampling planner has the best performance when the sample radius is 32, but the adaptive sampling planner shows better performance with a larger radius; this indicates that our adaptive-sampling approach improves the performance by generating fewer samples. The result also shows that a too-small sample radius decreases the planning performance due to the exponential increase in the number of samples.

C. Scalability Comparison Among Planners

In order to evaluate the scalability of the planners, we show the speedup of the parallel Poisson-RRT algorithm based on increasing the number of threads on a 24-core (48 threads) workstation in Fig. 8. Table IV shows the planning time for both CPU- and GPU-based Poisson-RRT planners with different number of threads. Our Poisson-RRT algorithm shows near-linear speedup as the number of parallel threads increases for both CPU and GPU versions in the common thread number range (∼16) of commodity PCs, while pRRT (AND parallel RRT) does not. At the large numbers (>24) of threads, the CPU-based planner is affected by the thread synchronization and has a sublinear performance. However, the GPU-based planner maintains the scalability (see the result of AlphaPuzzle in Table IV) with 32 threads.

There are several parallel RRT algorithms that can also achieve linear speedups on different computing systems. Jacobs *et al.* [20] use configuration space subdivision and parallel nearest neighbor search. This algorithm also includes a technique to balance the load between local computation and global computation for distributed systems. Ichnowski and Alterovitz [30] use a similar approach, partition-based sampling on shared-memory systems. In order to reduce the synchronization overhead, they use lock-free data structures. However, none of these methods have been evaluated on commodity manycore GPUs. As described in [30], these algorithms does not map well to single-instruction multiple-data (SIMD) GPU architectures. In partition-based algorithms, node expansions avoid generation of redundant close nodes by generating new nodes in separated regions of the configuration space, but it also makes the SIMD execution of multiple expansions less efficient due to lack of locality. There are tree-based planning approaches that use GPUs [35], but the parallelization in these algorithms is only limited to collision checking, and not the entire tree expansion, which requires extra CPU–GPU data transmission for each iteration.

Our `extendMPDS()` procedure maps well to SIMD-based GPU architectures. The use of precomputed Poisson-disk samples allows efficient expansion of the tree on GPUs without generating redundant nodes. It also exploits the parallel nearest neighbor search (see Section V-B) and collision checking (see Section V-C).

Furthermore, the use of precomputed samples allows our Poisson-RRT to outperform the original RRT algorithm in some complex benchmarks, even for single-threaded cases. As discussed in Section VII-A, the precomputed samples reduce the number of samples generated at runtime and improve the overall performance.

On the other hand, the results in Table II show that the performance improvement from the use of Poisson-disk sampling for some benchmark (e.g., Alpha Puzzle) is less than others. As can be inferred from the results in Table III, the speedup mainly comes from the exploiting the precomputed samples. If the number of precomputed samples is inadequate for the complexity of the benchmark scenarios, it causes additional runtime adaptive sample generations, which degenerate the planning performance. However, in such complex environments, the Poisson-RRT shows better performance than the planner with grid-based sampling, which is worse than the random sampling-based planner.

Fig. 9 shows the timing breakdown of the parallel planning algorithms corresponding to pRRT and parallel Poisson-RRT. The percentage of time spent in nearest neighbor computation is reduced in Poisson-RRT computation, as it exploits the maximal properties of Poisson-disk samples. On the other hand, nearest neighbor computation takes a higher fraction of total time in

TABLE IV
PERFORMANCE OF PARALLEL POISSON-RRT ALGORITHMS WITH DIFFERENT NUMBER OF THREADS

Algorithm	CPU-based Parallel Poisson-RRT				GPU-based Parallel Poisson-RRT					
	Time(s)									
Threads	1	2	4	8	1	2	4	8	16	32
Easy	0.35	0.26	0.12	0.18	0.43	0.25	0.12	0.06	0.05	0.03
Cubicle	2.59	1.31	0.85	0.59	4.00	2.96	1.78	1.21	0.63	0.31
AlphaPuzzle	27.15	15.42	6.51	6.69	47.24	21.69	12.19	5.79	2.67	1.38
Apartment	72.54	38.97	30.81	18.80	61.50	29.04	26.12	20.79	13.63	11.88

We report the planning time for each benchmark case.

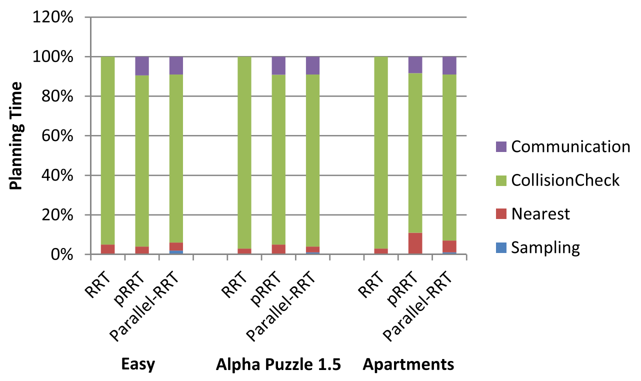
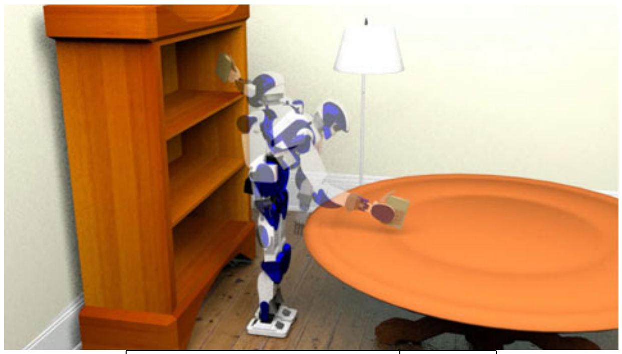


Fig. 9. Timing breakdown among various components for RRT, pRRT, and parallel Poisson-RRT algorithms for different benchmarks.



RRT (Single CPU Core)	6.17s
GPU Poisson-RRT	0.32s
Speed up	19.28x

Fig. 10. Motion planning of the 23-DOF HRP-4 robot using the parallel Poisson-RRT algorithm.

pRRT, and this `nearestNode` computation is a major source of inefficiency for pRRT.

D. High-DOF Robot Planning

Sampling-based planning algorithms like RRT are preferred for high-DOF planning, because they do not suffer from the curse of dimensionality. As described in Section II-C, there are no practical algorithms for computing maximal Poisson-disk samples in high-dimensional spaces, but relaxed Poisson-disk sampling algorithm [38] can be used to generate appropriate samples.

We evaluate our GPU-based parallel Poisson-RRT algorithm with the precomputed relaxed Poisson-disk samples for planning of HRP-4 robot in a simulation environment shown in Fig. 10. The environment has several static obstacles, and a book is attached to the right hand of the robot. We compute a collision-free motion for given initial and goal poses of the upper body of HRP-4 robot, which has 23 DOFs. We assume that the lower body of the robot is fixed and does not consider the dynamics constraints of the robot.

We measure the planning time of the GPU-based parallel Poisson-RRT with respect to single-core CPU algorithm. As shown in the table in Fig. 10, our parallel Poisson-RRT algorithm computes a collision-free path in real time using a precomputed relaxed Poisson-disk sample set, while the single-core CPU RRT takes several seconds to find a solution. It shows that our parallel Poisson-RRT algorithm can improve the performance of the planner for high-DOF scenarios.

VIII. LIMITATIONS, CONCLUSIONS, AND FUTURE WORK

In this paper, we have presented a new RRT-based motion planning algorithm based on Poisson-disk sampling. It uses an adaptive maximal Poisson-disk sampling approach to reduce the number of nodes in the resulting tree and explore the free space. Our algorithm is based on the RRT motion-planning algorithm and exploits the multiple cores on GPUs.

Our algorithm has some limitations. The maximal Poisson-disk sampling algorithm that we used may require a large amount of memory to execute its precomputation step in high-dimensional spaces, especially when r is small. Our current formulation takes into account only collision-free constraints, not nonholonomic or dynamic constraints. We only observe good speedups in challenging scenarios and in the parallel version of the algorithm.

There are many avenues for future work. The performance of our planning algorithm can be considerably improved by various optimizations, used for adaptive sampling or tree expansions, including bidirectional search similar to that used by RRT-Connect. The performance of our CPU-based parallel planner on workstations or clusters with very large numbers of CPU cores can be improved using lock-free data structures [30]. We would like to investigate techniques for automatically computing the optimal r for Poisson-disk sampling, for varying configuration space boundaries of higher dimensional problems. It would also be useful to take into account nonholonomic constraints.

References

[1] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, 1996.

[2] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," *Int. J. Robot. Res.*, vol. 20, no. 5, pp. 378–400, 2001.

[3] R. L. Cook, "Stochastic sampling and distributed ray tracing," in *An Introduction to Ray Tracing*. New York, NY, USA: Academic, 1989, pp. 161–199.

[4] A. Glassner, *An Introduction to Ray Tracing*. Burlington, MA, USA: Morgan Kaufmann, 1989.

[5] A. Lagae and P. Dutré, "A comparison of methods for generating Poisson disk distributions," *Comput. Grap. Forum*, vol. 27, no. 1, pp. 114–129, 2008.

[6] C. Park, J. Pan, and D. Manocha, "Poisson-RRT," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2014, pp. 4667–4673.

[7] J. Kuffner Jr., and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proc. Int. Conf. Robot. Autom.*, 2000, vol. 2, pp. 995–1001.

[8] A. Yershova, L. Jaillet, T. Simon, and S. M. LaValle, "Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain," in *Proc. Int. Conf. Robot. Autom.*, 2005, pp. 3867–3872.

[9] S. Rodriguez, X. Tang, J.-M. Lien, and N. M. Amato, "An obstacle-based rapidly-exploring random tree," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2006, pp. 895–900.

[10] R. Diankov, N. Ratliff, D. Ferguson, S. Srinivasa, and J. Kuffner, "Bispace planning: Concurrent multi-space exploration," in *Proc. Robot.: Sci. Syst.*, 2008.

[11] S. Rodriguez, S. Thomas, R. Pearce, and N. M. Amato, "RESAMPL: A region-sensitive adaptive motion planner," in *Algorithmic Foundation of Robotics VII*. Berlin, Germany: Springer, 2008, pp. 285–300.

[12] A. Shkolnik and R. Tedrake, "Sample-based planning with volumes in configuration space," *CoRR*, vol. abs/1109.3145, 2011.

[13] J. Denny, M. Morales, S. Rodriguez, and N. M. Amato, "Adapting RRT growth for heterogeneous environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2013, pp. 1772–1778.

[14] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, 2011.

[15] O. Arslan and P. Tsiotras, "Use of relaxation methods in sampling-based algorithms for optimal motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 2421–2428.

[16] O. Salzman and D. Halperin, "Asymptotically near-optimal RRT for fast, high-quality, motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2014, pp. 4680–4685.

[17] T. Lozano-Pérez and P. A. O'Donnell, "Parallel robot motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1991, pp. 1000–1007.

[18] N. M. Amato and L. K. Dale, "Probabilistic roadmap methods are embarrassingly parallel," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1999, vol. 1, pp. 688–694.

[19] R. Brooks and T. Lozano-Pérez, "A subdivision algorithm in configuration space for findpath with rotation," *IEEE Trans. Syst. Man, Cybern.*, vol. SMC-15, no. 2, pp. 224–233, Mar./Apr. 1985.

[20] S. Jacobs, K. Manavi, J. Burgos, J. Denny, S. Thomas, and N. Amato, "A scalable method for parallelizing sampling-based motion planning algorithms," in *Proc. Int. Conf. Robot. Autom.*, 2012, pp. 2529–2536.

[21] S. A. Jacobs, N. Stradford, C. Rodriguez, S. Thomas, and N. M. Amato, "A scalable distributed RRT for motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 5088–5095.

[22] C. Rodriguez, J. Denny, S. A. Jacobs, S. Thomas, and N. M. Amato, "Blind RRT: A probabilistically complete distributed RRT," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2013, pp. 1758–1765.

[23] E. Plaku and L. Kavraki, "Distributed sampling-based roadmap of trees for large-scale motion planning," in *Proc. Int. Conf. Robot. Autom.*, 2005, pp. 3868–3873.

[24] D. Devaurs, T. Siméon, and J. Cortés, "Parallelizing RRT on distributed-memory architectures," in *Proc. Int. Conf. Robot. Autom.*, 2011, pp. 2261–2266.

[25] M. Otte and N. Correll, "Path planning with forests of random trees: Parallelization with super linear speedup," *Dept. Comput. Sci., Univ. Colorado, Boulder, CO, USA, Tech. Rep. CU-CS* 1079–11, 2011.

[26] B. Raveh, A. Enosh, and D. Halperin, "A little more, a lot better: Improving path quality by a path-merging algorithm," *IEEE Trans. Robot.*, vol. 27, no. 2, pp. 365–371, Apr. 2011.

[27] S. Carpin and E. Pagello, "On parallel RRTs for multi-robot systems," in *Proc. Italian Assoc. Artif. Intell.*, 2002, pp. 834–841.

[28] I. Aguinaga, D. Borro, and L. Matey, "Parallel RRT-based path planning for selective disassembly planning," *Int. J. Adv. Manuf. Technol.*, vol. 36, no. 11, pp. 1221–1233, 2008.

[29] I. Sucan and L. E. Kavraki, "A sampling-based tree planner for systems with complex dynamics," *IEEE Trans. Robot.*, vol. 28, no. 1, pp. 116–131, Feb. 2012.

[30] J. Ichnowski and R. Alterovitz, "Parallel sampling-based motion planning with superlinear speedup," in *Proc. IEEE/RSJ Intell. Robots Syst.*, 2012, pp. 1206–1212.

[31] C. Pisula, K. Hoff, M. Lin, and D. Manocha, "Randomized path planning for a rigid body based on hardware accelerated Voronoi sampling," in *Proc. Workshop Algorithmic Found. Robot.*, vol. 18, pp. 279–292, 2000.

[32] J. Pan, C. Lauterbach, and D. Manocha, "g-Planner: Real-time motion planning and global navigation using GPUs," in *Proc. AAAI Conf. Artif. Intell.*, 2010.

[33] J. T. Kider, M. Henderson, M. Likhachev, and A. Safonova, "High-dimensional planning on the GPU," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2010, pp. 1245–1251.

[34] C. Park, J. Pan, and D. Manocha, "Real-time optimization-based planning in dynamic environments using GPUs," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 4090–4097.

[35] J. Bialkowski, S. Karaman, and E. Frazzoli, "Massively parallelizing the RRT and the RRT*," in *Proc. Int. Conf. Intell. Robots Syst.*, 2011, pp. 3513–3518.

[36] A. Lagae and P. Dutré, "A procedural object distribution function," *ACM Trans. Grap.*, vol. 24, no. 4, pp. 1442–1461, 2005.

[37] M. Ebeida, S. Mitchell, A. Patney, A. Davidson, and J. Owens, "A simple algorithm for maximal Poisson-disk sampling in high dimensions," *Comput. Grap. Forum*, vol. 31, no. 2, pp. 785–794, 2012.

[38] M. S. Ebeida *et al.*, "Spoke darts for efficient high dimensional blue noise sampling," *CoRR*, vol. abs/1408.1118, 2014. [Online]. Available: http://arxiv.org/abs/1408.1118

[39] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006.

[40] H. Niederreiter, *Quasi-Monte Carlo Methods*. Hoboken, NJ, USA: Wiley, 1992.

[41] R. Bohlin, "Path planning in practice; lazy evaluation on a multi-resolution grid," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2001, vol. 1, pp. 49–54.

[42] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *Int. J. Robot. Res.*, vol. 28, no. 8, pp. 933–945, 2009.

[43] M. Pivtoraiko and A. Kelly, "Differentially constrained motion replanning using state lattices with graduated fidelity," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2008, pp. 2611–2616.

[44] E. Kushilevitz, R. Ostrovsky, and Y. Rabani, "Efficient search for approximate nearest neighbor in high dimensional spaces," *SIAM J. Comput.*, vol. 30, no. 2, pp. 457–474, 2000.

[45] G. Marsaglia *et al.*, "Choosing a point from the surface of a sphere," *Ann. Math. Statist.*, vol. 43, no. 2, pp. 645–646, 1972.

[46] V. Garcia, E. Debreuve, and M. Barlaud, "Fast k nearest neighbor search using GPU," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Workshops*, 2008, pp. 1–6.

[47] J. Pan, C. Lauterbach, and D. Manocha, "Efficient nearest-neighbor computation for GPU-based motion planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2010, pp. 2243–2248.

[48] S. Gottschalk, M. C. Lin, and D. Manocha, "OBBTree: A hierarchical structure for rapid interference detection," in *Proc. 23rd Annu. Conf. Comput. Grap. Interact. Techn.*, 1996, pp. 171–180.

[49] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha, "Fast BVH construction on GPUs," *Comput. Grap. Forum*, vol. 28, no. 2, pp. 375–384, 2009.

[50] I. A. Şucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robot. Autom. Mag.*, vol. 19, no. 4, pp. 72–82, Dec. 2012. [Online]. Available: http://ompl.kavrakilab.org.

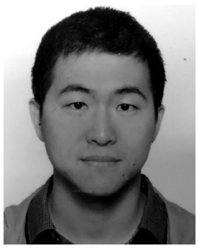
[51] C. Park, J. Pan, and D. Manocha, "Parallel RRT using Poisson-disk sampling," *Dept. Comput. Sci., Univ. North Carolina, Chapel Hill, NC, USA, Tech. Rep.*, 2013.

[52] R. Bohlin and L. Kavraki, "Path planning using lazy PRM," in *Proc. Int. Conf. Robot. Autom.*, vol. 1, 2000, pp. 521–528.

[53] J. M. Hammersley, "Monte carlo methods for solving multivariable problems," *Ann. New York Acad. Sci.*, vol. 86, no. 3, pp. 844–874, 1960.

Chonhyon Park (S'13) received the B.S. and M.S. degrees in computer science and engineering from Seoul National University, Seoul, South Korea, in 2005 and 2007, respectively. He is currently working toward the Ph.D. degree with the Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA.

He was an Intern with the Honda Research Institute, Mountain View, CA, USA; Samsung Research America, San Jose, CA; and Disney Research, Glendale, CA. His research interests include motion and path planning, navigation of virtual characters, and manycore computing.



Jia Pan (M'15) received the B.E. degree in automation from the Department of Automation, Tsinghua University, Beijing, China, in 2005; the M.S. degree in pattern recognition and intelligent systems from the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing, in 2008; and the Ph.D. degree in computer science from the University of North Carolina at Chapel Hill, NC, USA, in 2013.

He was a Postdoctoral Researcher with the Department of Electrical Engineering and Computer Science, University of California, Berkeley. He joined the faculty with the Department of Computer Science, University of Hong Kong, Hong Kong, in 2014 and then moved to the Department of Mechanical and Biomedical Engineering at the City University of Hong Kong as an assistant professor. His research interests include motion planning, general-purpose graphics processing unit, and machine learning for robotics.

Dinesh Manocha (F'11) received the Ph.D. degree in computer science from University of California at Berkeley, CA, USA, in 1992.

He is the Phi Delta Theta/Mason Distinguished Professor of computer science with University of North Carolina at Chapel Hill, NC, USA. Along with his students, he has also received 14 best paper awards at leading conferences. He has published more than 450 papers, as well as some of the software systems related to collision detection, graphics processing unit-based algorithms, and geometric computing developed by his group have been downloaded by more than 150 000 users and are widely used in the industry. He has supervised 30 Ph.D. dissertations.

Dr. Manocha is a Fellow of the Association for Computing Machinery and the American Association for the Advancement of Science. He received the Distinguished Alumni Award from the Indian Institute of Technology, Delhi.