2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

Poisson-RRT

Chonhyon Park and Jia Pan and Dinesh Manocha
http://gamma.cs.unc.edu/PoissonRRT/

Abstract— **We present an RRT-based 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. Our approach can be easily parallelized on multi-core CPUs and many-core GPUs. We highlight the performance of our algorithm on different benchmarks.**

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 serial and parallel algorithms.

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 [6], [7], [8]. Poisson-disk distributions are known to have good blue-noise characteristics and are widely used in statistics, computer graphics, mesh algorithms, AI, 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.

*This research is supported in part by ARO Contracts W911NF-10-1-0506 and NSF awards 1000579, 1117127, 1305286.

Chonhyon Park and Jia Pan and Dinesh Manocha are with the Department of Computer Science, University of North Carolina at Chapel Hill. E-mail: {chpark, panj, dm}@cs.unc.edu.

In this paper, we present a Poisson-RRT algorithm that uses Poisson-disk samples to generate an RRT tree. We use maximal Poisson-disk samples to compute the RRT tree in arbitrary dimensions. The nodes are computed based on these samples, and we use the *free-disk property* of Poisson-disk samples for tree expansion. Furthermore, we present an adaptive sampling scheme that increases the sampling rate in the challenging regions of the configuration space (e.g. narrow passages). We also use the Poisson-disk samples to parallelize the RRT algorithm. As compared to prior parallel-RRT algorithms, we show that Poisson-disk samples result in fewer redundant nodes in the configuration space and are more amenable to parallelization on commodity multi-core CPUs and many-core GPUs.

The rest of the 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 serial and parallel algorithms in Section IV. We highlight the performance on different benchmarks in Section V.

II. Related Work and Background

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

A. RRT-based Motion Planning

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

U.S. Government work not protected by
U.S. copyright

Parallel RRT Algorithms: Many parallel techniques have been proposed to improve the performance of RRT algorithms on multiple cores. At a broad level, these can be classified into AND parallelization and OR parallelization [15], [16], [17]. Many recent techniques exploit multiple CPU and GPU cores to parallelize collision checking or subdividing the configuration space [18], [3].

B. Maximal Poisson-Disk Sampling

Poisson-disk sampling [6], [7], [8] 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:

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

where $\mathbf{X} = \{\mathbf{x}_i\}$ is the set of samples in domain Ω. Given a non-maximal 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 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-D. The bias-free property of the Poisson-disk samples (which functions similarly to the Voronoi diagram bias used in the original RRT algorithm [9]).

Recently, many algorithms have been suggested for fast computation of Poisson-disk sampling, such as [19], in order to generate samples with linear time complexity. Parallel Poisson-disk sampling algorithms have also been designed for higher-dimensional spaces [20]. We can use any of these algorithms to compute maximal Poisson-disk samples in any dimension.

C. Lattice-based Sampling

Although random sampling is widely used in motion planning, many other sampling techniques have been proposed [21]. 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 [22]. Lattices are a generalization of grids that allow non-orthogonal 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. Multi-resolution approaches [23], [24] are used to increase the number of samples in lattice-based planning algorithms, and have been combined with replanning [25]. Like these lattice-based techniques, 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.

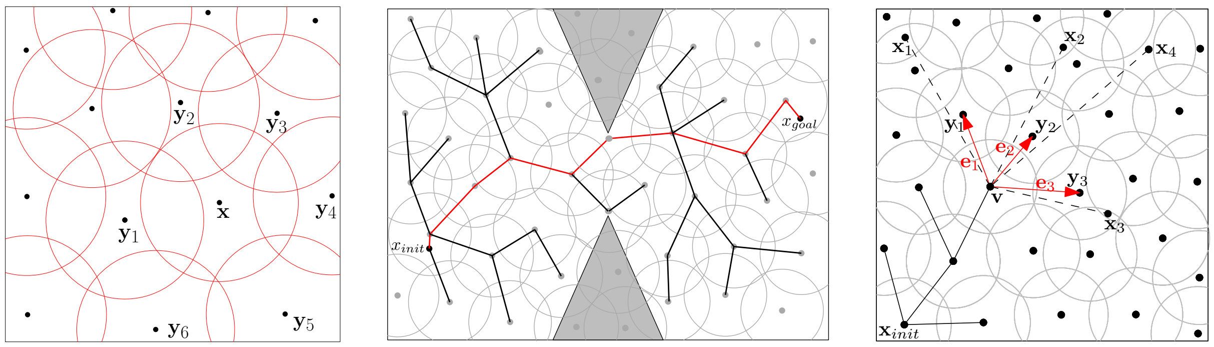
A. Assumptions and Notations

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

The RRT tree $\mathbf{T}$ is initialized with the root node of $\mathbf{x}_{init}$, 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 [26] with computational complexity $\mathcal{O}(d \log n)$ are used, where d is the dimension of the configuration space.

   The local planning step checks whether the shortest path between $\mathbf{v}$ and $\mathbf{x}$ lies in $\mathcal{C}_{free}$ (i.e., that the



(a) Maximal Poisson-disk samples (b) Motion planning using Poisson-disk sampling (c) Parallel Poisson-RRT tree expansion

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 $\mathbf{x}_{init}$ to $\mathbf{x}_{goal}$. (c) Parallel Poisson-RRT tree expansion using 4 threads. The i-th 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.

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 $\mathbf{x}_{new}$ 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.

Algorithm 1 RRT Planning using Poisson-disk Sampling

```
Require: start configuration $\mathbf{x}_{init}$, goal configuration $\mathbf{x}_{goal}$,
    precomputed Poisson-disk (radius $r$) sample set $\mathbf{X}$
Ensure: 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 /* For serial version, m=1 */
5:     while $\mathbf{x}_{goal} \notin \mathbf{T}$ do
6:         /* Pick a random point within a random selected Poisson-disk */
7:         $\mathbf{x} \leftarrow$ sample()
8:         $\mathbf{T} \leftarrow$ extend($\mathbf{T}, \mathbf{x}, \mathbf{X}$)
9: end for
```

The overall Poisson-RRT algorithm is shown in Algorithm 1. 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. Unlike the standard RRT, which performs local planning between the nearest node $\mathbf{v}$ and the configuration $\mathbf{x}$, our algorithm chooses a Poisson-disk sample $\mathbf{x}_{nbr}$ 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 $\mathbf{x}_{nbr}$ and $\mathbf{v}$, $\mathbf{x}_{nbr}$ is added to the RRT tree as a new node. The tree expansion is repeated until the goal configuration $\mathbf{x}_{goal}$ 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. 2 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.

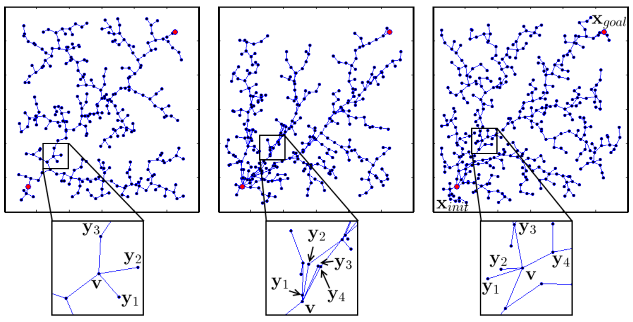
IV. POISSON-RRT ALGORITHM

In this section, we present the details of our serial and parallel planning algorithms, including precomputation of maximal Poisson-disk samples, tree expansion, and adaptive sampling. The analysis of the algorithm can be found in [27].

A. Precomputation of Maximal Poisson-disk Samples

As a precomputation step, Poisson-disk samples are generated in the d-dimensional configuration space. These samples are independent of obstacles, so we can use a precomputed sample set computed offline for multiple planning queries.

Our Poisson-disk sample generation is based on the fast algorithm proposed by Ebeida et al. [28]. For a given disk



(a) RRT (b) Parallel RRT (c) Parallel Poisson-RRT

Fig. 2: Comparison of RRT trees generated using different planning approaches. (a) The tree corresponding to the original RRT algorithm is generated according to the Voronoi bias of the sequential algorithm. (b) The 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) The 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.

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.

B. Tree Expansion

Given a new random sample $\mathbf{x}$, our algorithm extends the planning tree $\mathbf{T}$ using the `extend` procedure, which is summarized in Algorithm 2.

For a sample point $\mathbf{x}$, the algorithm finds the nearest node $\mathbf{v}$ in $\mathbf{T}$. From a node $\mathbf{v}$, The Poisson-RRT algorithm chooses a sample $\mathbf{x}_{nbr}$, which is a point closest to $\mathbf{x}$ among $\mathbf{v}$'s neighboring Poisson-disk samples. These steps utilize the nearest neighbor search. There has been extensive work done on the nearest neighbor search using GPUs [29], [30], [5]. We use the algorithm proposed by Pan et al. [30], 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.

It is possible that a Poisson-disk sample can be chosen by more than one thread in the nearest neighbor search (line 1-2). 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.

C. 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 [31] for the triangle model representations of the robot and obstacles using a GPU-based construction algorithm [32]. The OBB trees improve the performance of collision checking because of their high culling efficiency.

Algorithm 2 RRT tree `extend()` procedure using maximal Poisson-disk sampling.

```
Require: RRT Tree $\mathbf{T}$, a new random sample $\mathbf{x}$, Poisson-disk sample set $\mathbf{X}$
Ensure: 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:      $\mathbf{X}$.remove($\mathbf{x}_{nbr}$)
8:  else
9:      /* if there is collision, perform adaptive sampling */
10:     if $\rho(\mathbf{x}_{free}, \mathbf{v}) < \mathbf{v}.r$ then
11:         /* If the collision occurs in the disk of $\mathbf{v}$, reduce the coverage of $\mathbf{v}$
12:         $\mathbf{v}.r \leftarrow \mathbf{v}.r/2$
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:     $\mathbf{X}$.add(adaptiveSampling($\mathbf{v}, \mathbf{x}_{nbr}, \mathbf{x}_{free}$))
```

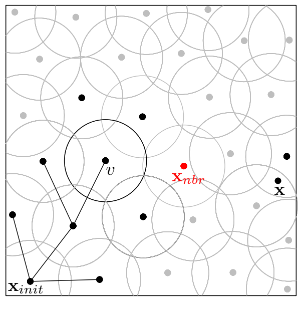
When the tree node and the nearest Poisson-disk sample are computed, the algorithm performs local planning to check for a feasible path between the two configurations. We use discrete collision detection (DCD), which discretizes the path between two configurations into multiple steps, between the robot and obstacles; we then check collisions for each step. The collision checking performed during local planning is regarded as the most time-consuming part of the overall algorithm.

D. Adaptive Sampling

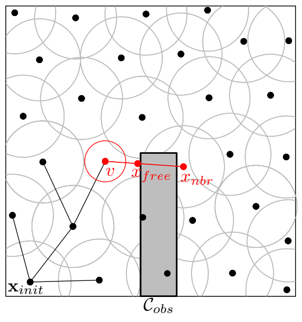
The precomputed Poisson-disk samples may not have a large-enough number of samples to find a collision-free solution. As a result, the algorithm performs adaptive Poisson subsampling at runtime to generate more samples with reduced distance between them. We perform adaptive sampling in the regions where the local planning routine finds a collision between an edge of the tree and an obstacle; in that sub-region of the configuration space, we generate samples with reduced disk radii.

In the precomputation step, we compute a template from the Poisson-disk sample set, which we use for adaptive sampling.. There is a sample with radius $r/2$ placed at the origin, and the algorithm randomly generates more samples to ensure that the new samples satisfy the maximal property in the disk of radius r.

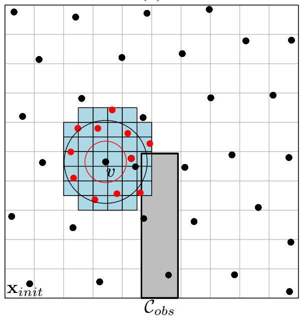
The `collisionCheck` procedure, used for local planning, checks for collisions along the edge that joins $\mathbf{v}$



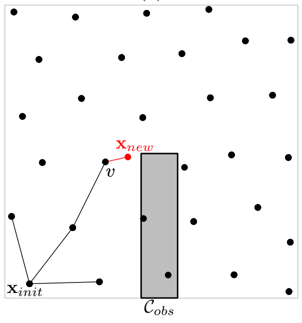
(a)



(b)



(c)



(d)

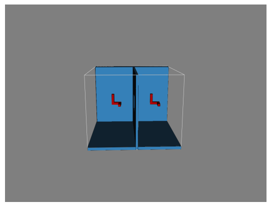
Fig. 3: Tree extension and adaptive sampling. (a) The sample $\mathbf{x}_{nbr}$ is the point closest to $\mathbf{x}$ among the neighboring Poisson-disk samples of $\mathbf{v}$. (b) If $\overline{\mathbf{v}\mathbf{x}_{nbr}}$ intersects $\mathcal{C}_{obs}$, `collisionCheck` procedure returns the last collision-free point $\mathbf{x}_{free}$. If the collision occurs within the disk associated with $\mathbf{v}$, the radius of this disk is reduced by half. (c) A precomputed template of Poisson-disk samples is applied to $\mathbf{v}$ to find a point which is close to $\mathbf{x}_{free}$ and satisfies the maximal property in the disk of $\mathbf{v}$. (d) A new sample $\mathbf{x}_{new}$ is added to $\mathbf{X}$; it can be connected to $\mathbf{T}$ if there is no collision on the local path joining $\mathbf{v}$ and $\mathbf{x}_{new}$.

to $\mathbf{x}_{nbr}$. If there is no collision, $\mathbf{x}_{nbr}$ is added to the tree and removed from $\mathbf{X}$. If a collision is detected, the procedure computes $\mathbf{x}_{free}$ as the last collision-free point on the direction from $\mathbf{v}$ to $\mathbf{x}_{nbr}$ (Fig. 3(b)). If a collision occurs in the disk associated with $\mathbf{v}$ or $\mathbf{x}_{nbr}$, the adaptive sampling algorithm reduces the radius of the disk by half. After this reduction, some regions that were covered in the original disk may now be uncovered, so we use the precomputed template to generate new samples in the region (Fig. 3(c)). Using the positions of $\mathbf{v}$ and $\mathbf{x}_{free}$, we compute which sample in the template is closest to $\mathbf{x}_{free}$ when the template is applied to $\mathbf{v}$. The new sample is connected to $\mathbf{T}$ and added to $\mathbf{X}$ for future expansion (Fig. 3(d)). The template is scaled to generate samples with different Poisson-disk radii during this adaptive sampling step.

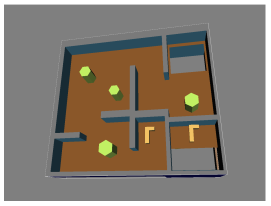
This adaptive sampling approach allows the algorithm to handle any width of narrow passages, since it adaptively generates more samples in the difficult regions of the configuration space.

V. Results

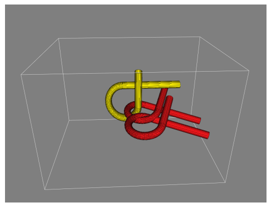
In this section, we present our experimental results and highlight the performance of our serial and parallel algorithms on different benchmarks. We implemented the algorithm using OMPL [33] and NVIDIA CUDA libraries. All the timings described in this section were generated on a commodity PC with an Intel i7-2600 8-core CPU and a NVIDIA GTX 680 GPUs. The details of parallel version of our algorithm and its implementation on multi-core CPUs and many-core GPUs are given in [27].



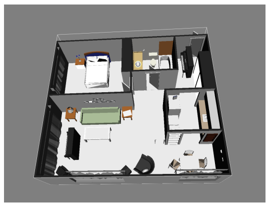
(a) Easy



(b) Cubicle



(c) Alpha Puzzle



(d) Apartment

Fig. 4: The 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 narrow passage; Apartment moves the piano to the hallway near the door entrance.

For the first experiment, we used four well-known benchmark scenarios from OMPL, shown in Fig. 4. These planning problems vary; 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 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 [27]:

- Standard RRT (RRT-Extend) [2] : Sequential RRT that uses random uniform sampling.
- RRT-Connect [9] : Bidirectional algorithm that expands trees from both the initial and the goal configurations.
- Lazy-RRT [34] : Algorithm that defers collision checks until it finds a solution.
- pRRT [17] : AND parallel RRT algorithm.

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. 5 shows the parallel algorithm's planning-time speedup

	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	191.79*	(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)

TABLE I: Performance of RRT-based planning algorithms on different benchmarks. We report planning time for each case. The mean and standard deviation are computed from 100 trials on each benchmark. CPU-based pRRT utilizes 8 threads to fully exploit the 8-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.

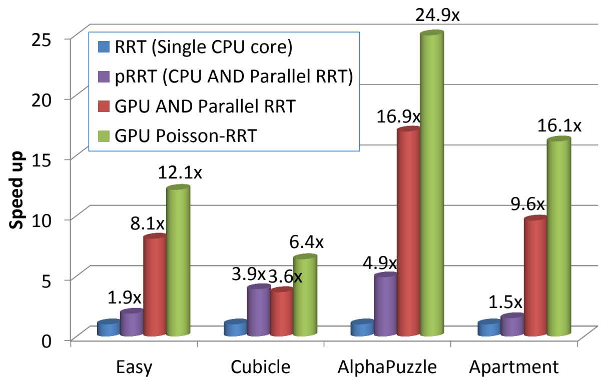


Fig. 5: Speedup of GPU-based algorithms from the original RRT algorithm, which uses a single CPU core. GPU-based Poisson-RRT improves the performance of CPU-based algorithm up to 24 times.

	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

TABLE II: Performance of Poisson-RRT algorithm with different sample radii for 'Easy' benchmark (Fig. 4(a)). 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.

on the OPML benchmarks as compared to the original CPU-based RRT algorithm, which uses a single core. In general, our GPU-based Poisson-RRT is faster than CPU-based algorithms, providing up to 25X speedup over the original RRT algorithm [27].

In the next experiment, 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' (Fig. 4(a)) is shown in Table II. The uniform sampling planner has the best performance when the sample radius is 32, but the adaptive sampling planner shows better performance with bigger radii; 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.

VI. 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 non-holonomic 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, including bidirectional search similar to that used by RRT-Connect. We would like to investigate techniques for automatically computing the optimal r for Poisson-disk sampling, especially for higher-dimensional problems. It would be useful to take into account non-holonomic constraints.

References

[1] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.
[2] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.
[3] S. Jacobs, K. Manavi, J. Burgos, J. Denny, S. Thomas, and N. Amato, "A scalable method for parallelizing sampling-based motion planning algorithms," in *International Conference on Robotics and Automation*, 2012, pp. 2529–2536.
[4] E. Plaku and L. Kavraki, "Distributed sampling-based roadmap of trees for large-scale motion planning," in *International Conference on Robotics and Automation*, 2005, pp. 3868–3873.
[5] J. Pan, C. Lauterbach, and D. Manocha, "g-planner: Real-time motion planning and global navigation using gpus," in *AAAI Conference on Artificial Intelligence*, 2010.
[6] R. L. Cook, "Stochastic sampling and distributed ray tracing," in *An introduction to ray tracing*. Academic Press Ltd., 1989, pp. 161–199.
[7] A. Glassner, *An introduction to ray tracing*. Morgan Kaufmann, 1989.
[8] A. Lagae and P. Dutré, "A comparison of methods for generating poisson disk distributions," in *Computer Graphics Forum*, vol. 27, no. 1. Wiley Online Library, 2008, pp. 114–129.

[9] J. Kuffner Jr and S. LaValle, “RRT-connect: An efficient approach to single-query path planning,” in *International Conference on Robotics and Automation*, vol. 2, 2000, pp. 995–1001.

[10] A. Yershova, L. Jaillet, T. Simon, and S. M. LaValle, “Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain,” in *International Conference on Robotics and Automation*, 2005, pp. 3867–3872.

[11] R. Diankov, N. Ratliff, D. Ferguson, S. Srinivasa, and J. Kuffner, “Bispace planning: Concurrent multi-space exploration,” in *Robotics: Science and Systems*, 2008.

[12] S. Rodriguez, S. Thomas, R. Pearce, and N. Amato, “Resampl: A region-sensitive adaptive motion planner,” in *Algorithmic Foundation of Robotics VII*, S. Akella, N. Amato, W. Huang, and B. Mishra, Eds., 2008, pp. 285–300.

[13] A. Shkolnik and R. Tedrake, “Sample-based planning with volumes in configuration space,” 2011, coRR, vol. abs/1109.3145.

[14] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[15] S. Carpin and E. Pagello, “On parallel rrts for multi-robot systems,” in *Italian Association for Artificial Intelligence*, 2002, pp. 834–841.

[16] I. Aguinaga, D. Borro, and L. Matey, “Parallel rrt-based path planning for selective disassembly planning,” *International Journal of Advanced Manufacturing Technology*, vol. 36, no. 11, pp. 1221–1233, 2008.

[17] D. Devaurs, T. Siméon, and J. Cortés, “Parallelizing rrt on distributed-memory architectures,” in *International Conference on Robotics and automation*, 2011, pp. 2261–2266.

[18] J. Bialkowski, S. Karaman, and E. Frazzoli, “Massively parallelizing the rrt and the rrt,” in *International Conference on Intelligent Robots and Systems*, 2011, pp. 3513–3518.

[19] A. Lagae and P. Dutré, “A procedural object distribution function,” *Transactions on Graphics*, vol. 24, no. 4, pp. 1442–1461, 2005.

[20] M. Ebeida, A. Davidson, A. Patney, P. Knupp, S. Mitchell, and J. Owens, “Efficient maximal poisson-disk sampling,” *Transactions on Graphics*, vol. 30, no. 4, p. 49, 2011.

[21] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[22] H. Niederreiter, *Quasi-Monte Carlo Methods*. Wiley Online Library, 1992.

[23] R. Bohlin, “Path planning in practice; lazy evaluation on a multi-resolution grid,” in *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, vol. 1. IEEE, 2001, pp. 49–54.

[24] M. Likhachev and D. Ferguson, “Planning long dynamically feasible maneuvers for autonomous vehicles,” *The International Journal of Robotics Research*, vol. 28, no. 8, pp. 933–945, 2009.

[25] M. Pivtoraiko and A. Kelly, “Differentially constrained motion replanning using state lattices with graduated fidelity,” in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. IEEE, 2008, pp. 2611–2616.

[26] E. Kushilevitz, R. Ostrovsky, and Y. Rabani, “Efficient search for approximate nearest neighbor in high dimensional spaces,” *SIAM Journal on Computing*, vol. 30, no. 2, pp. 457–474, 2000.

[27] C. Park, J. Pan, and D. Manocha, “Parallel RRT using Poisson-disk sampling,” Department of Computer Science, University of North Carolina at Chapel Hill, Tech. Rep., 2013.

[28] M. Ebeida, S. Mitchell, A. Patney, A. Davidson, and J. Owens, “A simple algorithm for maximal poisson-disk sampling in high dimensions,” *Computer Graphics Forum*, vol. 31, no. 2, pp. 785–794, 2012.

[29] V. Garcia, E. Debreuve, and M. Barlaud, “Fast k nearest neighbor search using GPU,” in *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW’08. IEEE Computer Society Conference on*. IEEE, 2008, pp. 1–6.

[30] J. Pan, C. Lauterbach, and D. Manocha, “Efficient nearest-neighbor computation for GPU-based motion planning,” in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, 2010, pp. 2243–2248.

[31] S. Gottschalk, M. C. Lin, and D. Manocha, “OBBTree: a hierarchical structure for rapid interference detection,” in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, 1996, pp. 171–180.

[32] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha, “Fast BVH construction on GPUs,” in *Computer Graphics Forum*, vol. 28, no. 2. Wiley Online Library, 2009, pp. 375–384.

[33] I. A. Şucan, M. Moll, and L. E. Kavraki, “The Open Motion Planning Library,” *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012, http://ompl.kavrakilab.org.

[34] R. Bohlin and L. Kavraki, “Path planning using lazy prm,” in *International Conference on Robotics and Automation*, vol. 1, 2000, pp. 521–528.