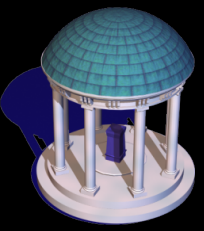
Real-time Collision Detection and Motion Planning in Dynamic Scenes

Dinesh Manocha
Department of Computer Science
UNC Chapel Hill

http://gamma.cs.unc.edu



Collaborators

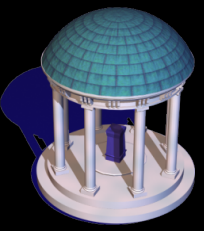
Jia Pan (UNC Chapel Hill)

Chonhyon Park (UNC Chapel Hill)

Christian Lauterbach (UNC Chapel Hill/Google)

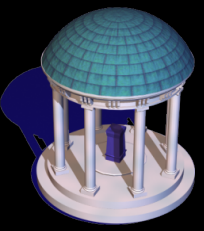
Sachin Chitta (Willow Garage)

Ioan Sucan (Willow Garage)



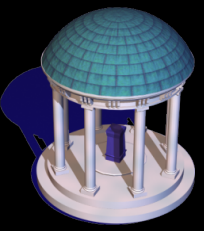
Talk Organization

- Collision Detection/Proximity Queries
- Motion Planning in Dynamic Environments



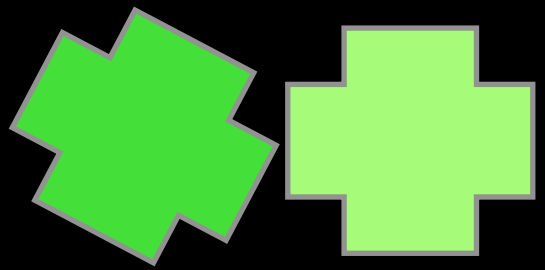
Talk Organization

- Collision Detection/Proximity Queries
- Motion Planning in Dynamic Environments

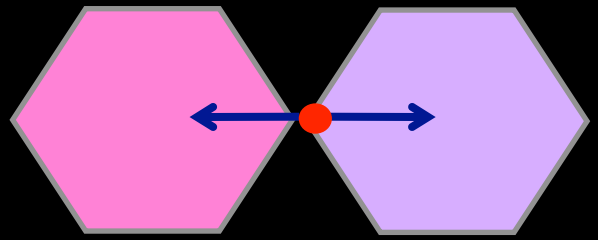


Collision & Proximity Queries

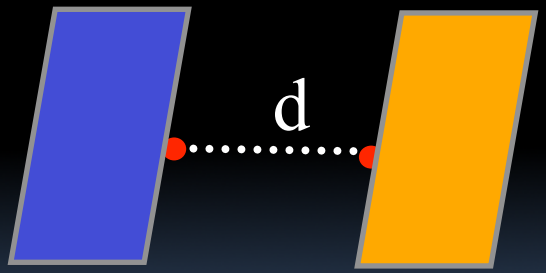
*Geometric reasoning of spatial relationships among objects (*in a dynamic environment)



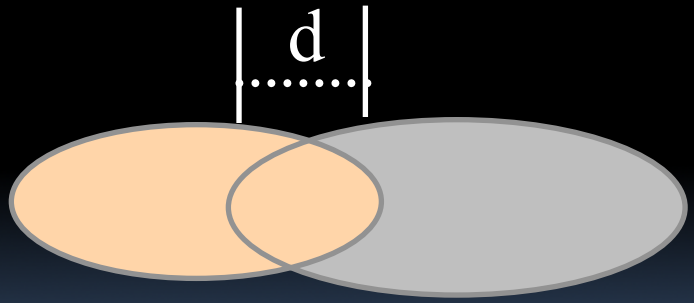
Collision Detection



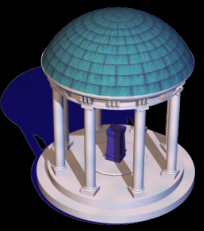
Contact Points & Normals



Closest Points & Separation Distance



Penetration Depth



Problem Domain Specifications

Model Representations

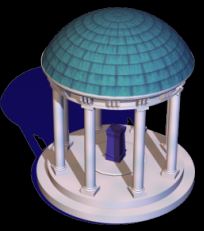
- polyhedra (convex vs. non-convex vs. soups)
- CSG, implicits, parametrics, point-clouds

Type of Queries

- discrete vs. continuous query
- distance vs. penetration computation
- estimated time to collision

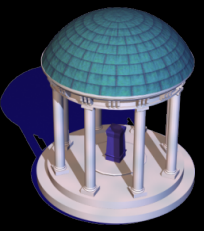
Simulation Environments

- pairwise vs. n-body
- static vs. dynamic
- rigid vs. deformable



Prior work on Proximity Computations

- Fast algorithms for convex polytopes (1991 onwards)
- Bounding volume hierarchies for general polygonal models (1995 onwards)
- Deformable models & self-collisions (2000 onwards)
- Use of GPUs and multi-core hardware (2005 onwards)



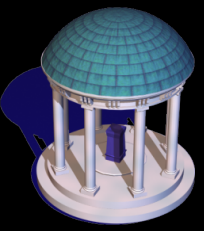
Prior work on Proximity Computations

Multiple software systems

- I-Collide, RAPID, PQP, DEEP, SWIFT, SWIFT++, DeformCD, PIVOT, Self-CCD,…..
- http://gamma.cs.unc.edu/software/#collision

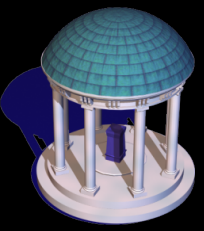
- More than 110,000 downloads from 1995 onwards

- Issued more than 55 commercial licenses (Kawasaki, MSC Software, Ford, Sensable, Siemens, BMW, Phillips, Intel, Boeing, etc.)

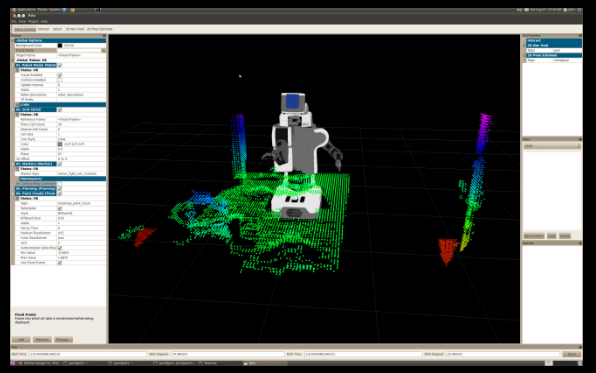


FCL: Motivation

- A new collision and proximity computation library
  - Flexible: different object types/ queries
  - Extensible: adding new algorithms is easy
  - Efficiency: similar performance with the best libraries
- Provide many functions from state-of-the-art research, more in future

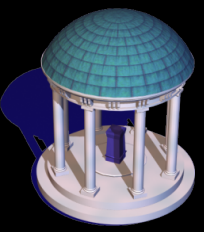


Robotics: Motivation

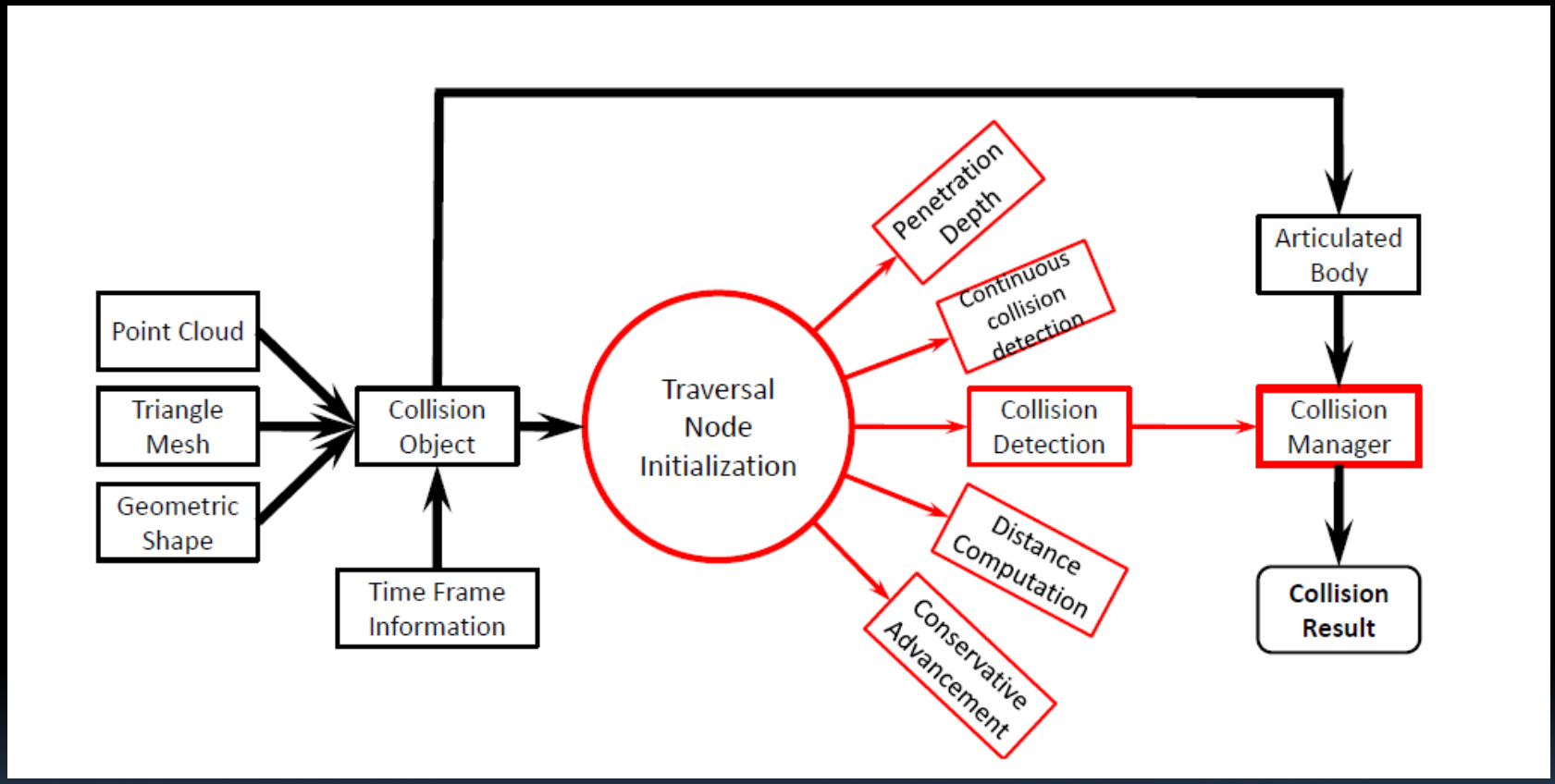


- Human environments
  - Clutter, dynamic obstacles
- Data from 3D sensors
  - Large number of points (~10k for laser scans, ~20k for stereo)
- Real-time computation important for fast online reactive grasping, motion planning
- Proximity computation important for many useful heuristics in robotics

Efficient collision and proximity computation is essential for any online robot operations in human environments

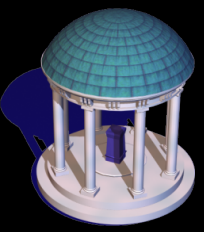


FCL Overview

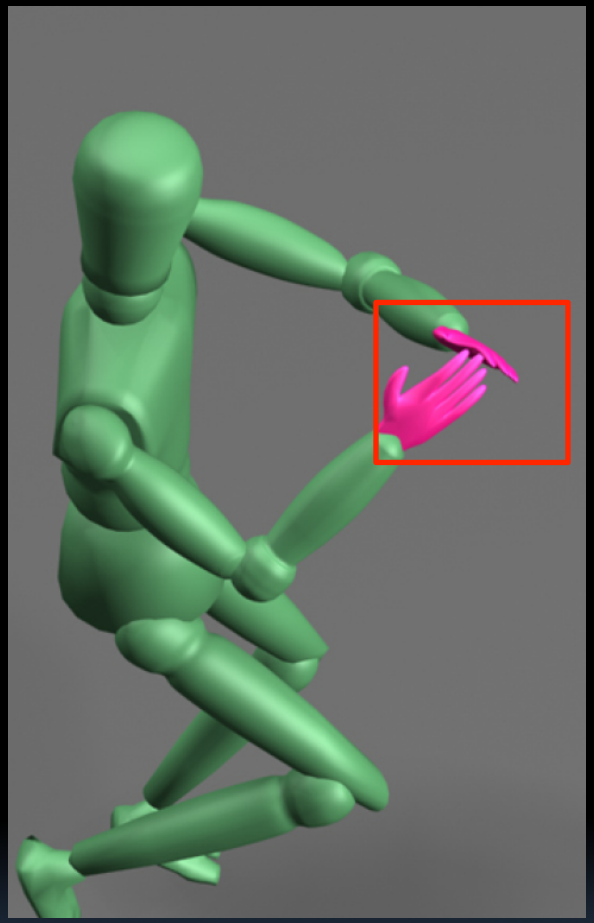


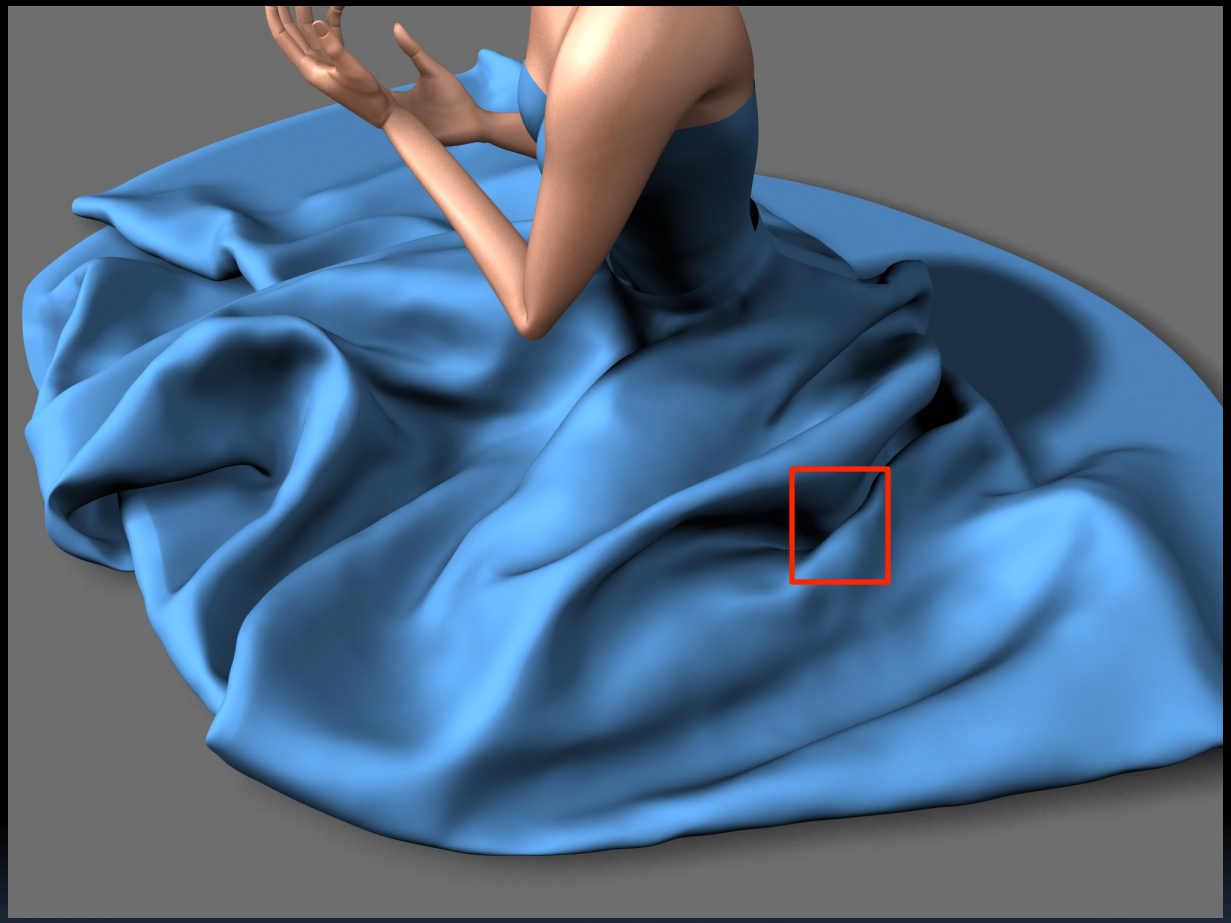
Supported Functions (Dec. 2012)

	Rigid Objects	Point Clouds	Deformable Objects	Articulated Objects
(Discrete) Collision Detection	Y	Y	Y	Y
Continuous Collision Detection	Y	Y	Y	Y
Self Collision Detection	Y	Y	Y	Y
Penetration Estimation	Y	N	N	N
Distance Computation	Y	N	Y	Y
Broad-phase Collision	Y	Y	Y	Y



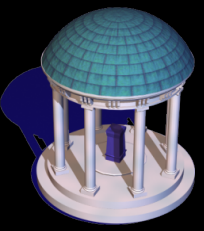
Challenging Scenarios





Articulated Model

Deformable Model



Download

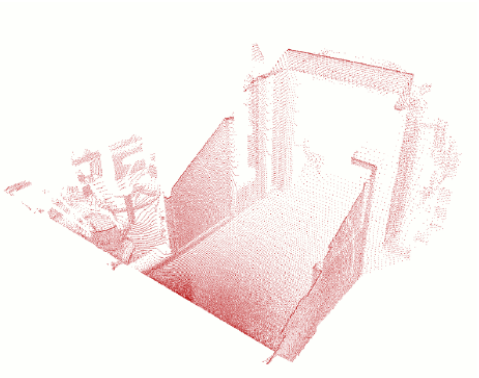
- Independent code, but ROS interface is provided
- Available at http://gamma.cs.unc.edu/FCL

Collision/Proximity Computation on Sensor Data

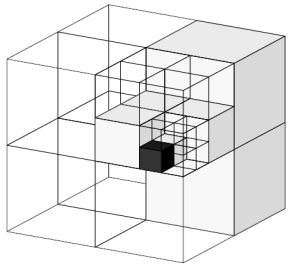
- Broad phase acceleration widely used for N-body cases

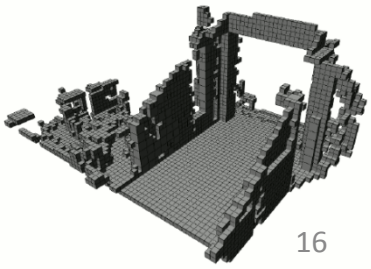
- Many real-world applications need to handle proximity computation for sensor data

Sensor Data



- Point cloud
  - Output from laser/Kinect, etc.
  - Cannot encode unknown regions
  - Very large
- Octree (octomap [Hornung et al. 2013])
  - Store point cloud in a compact manner
  - Support multi-resolution
  - Encode occupied/free/unknown regions



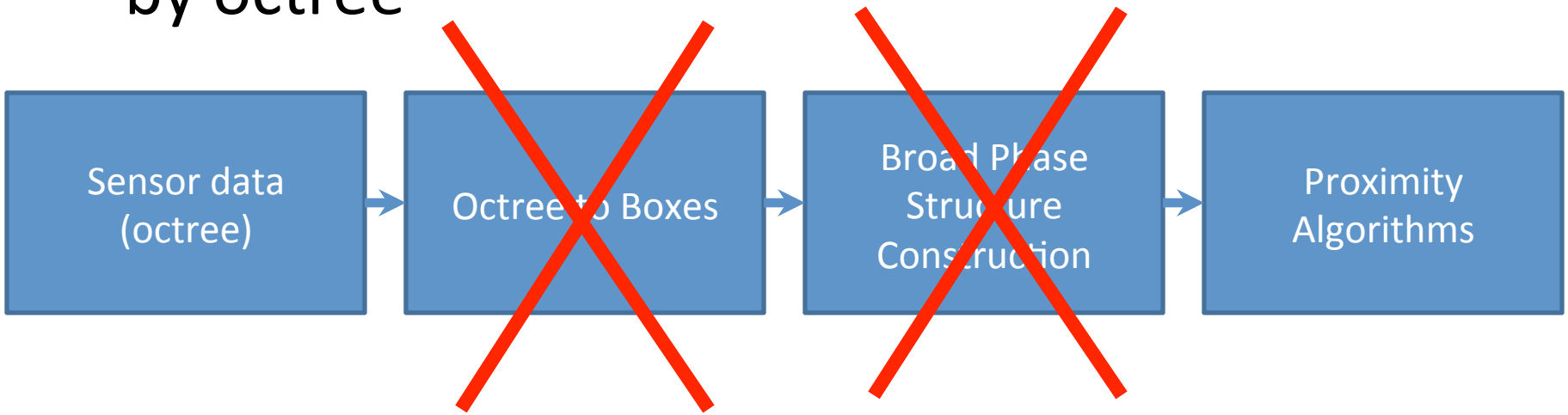


Broad Phase Algorithms

- The broad phase data structure is computed offline and its performance does not influence the online performance
  - The objects can be added/removed/moving
  - The broad phase data structure can be updated and reused for a long time

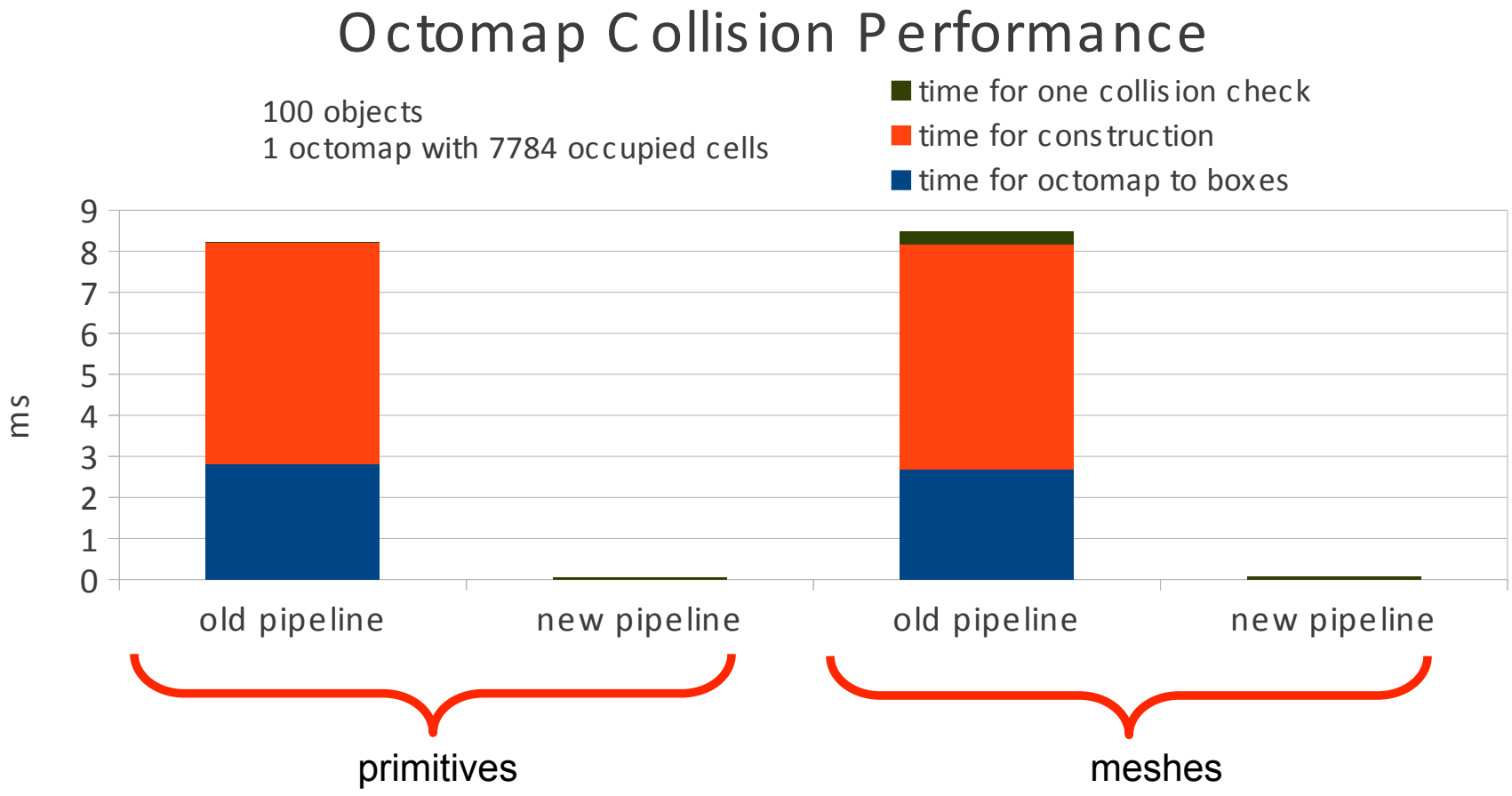
Our Solution: Completely Avoid Construction Overhead

- Directly collide with sensor data represented by octree



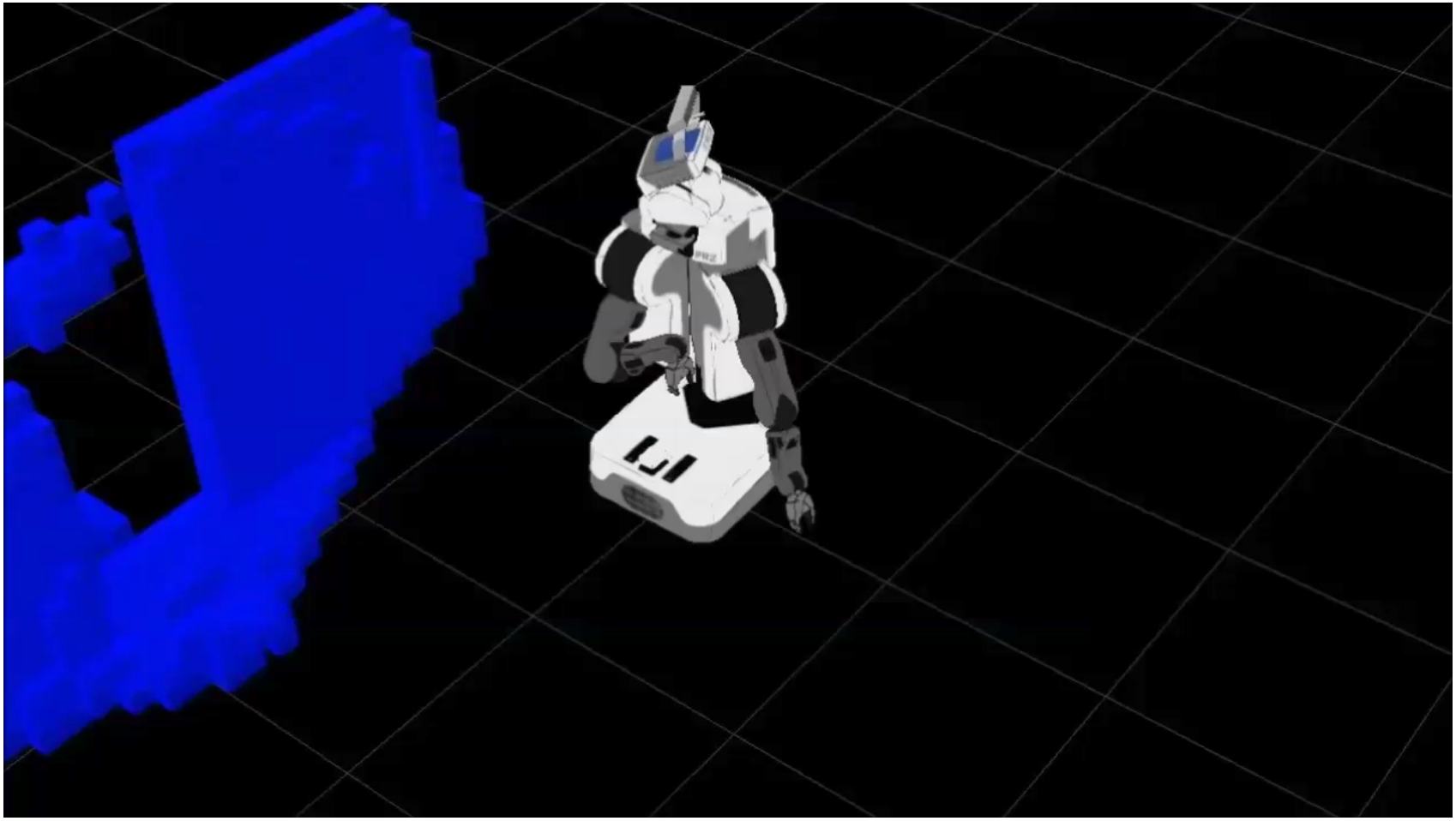
- Collision query time increases, but construction time is zero

Result



Implemented in MoveIt! (http://moveit.ros.org)

With Active Sensing

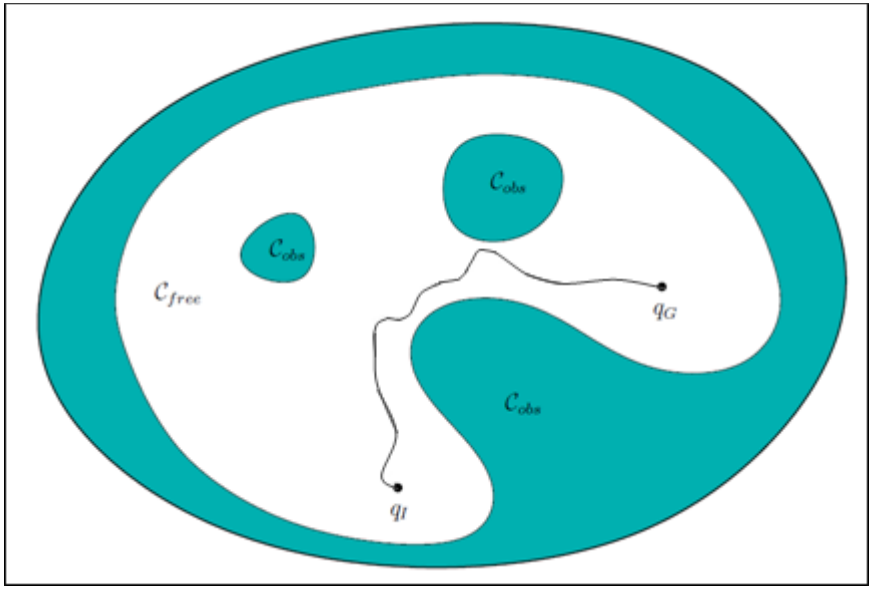


Talk Organization

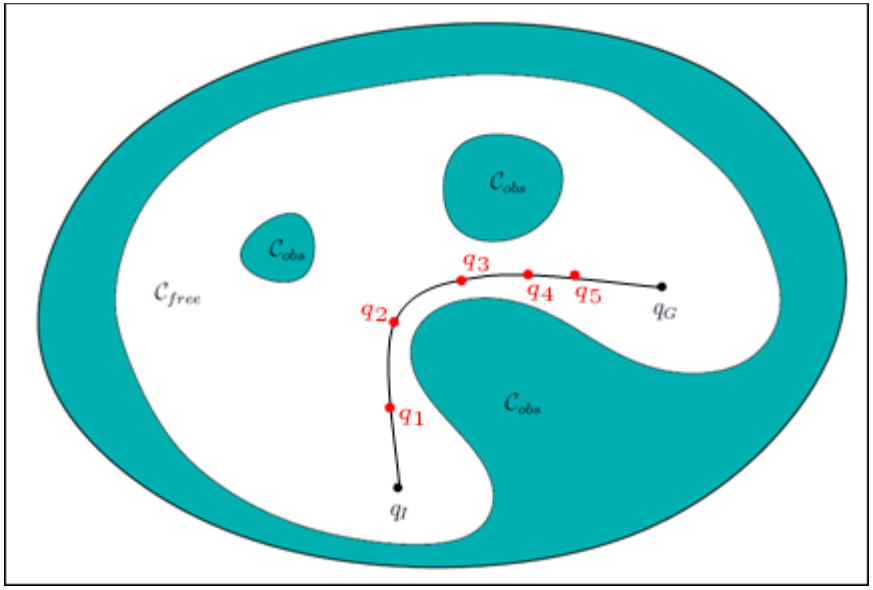
- Collision Detection/Proximity Queries
- Motion Planning in Dynamic Environments

Motion Planning Algorithms

- Random sampling-based algorithms



- Optimization-based algorithms



Dynamic Scenes: Our Solutions

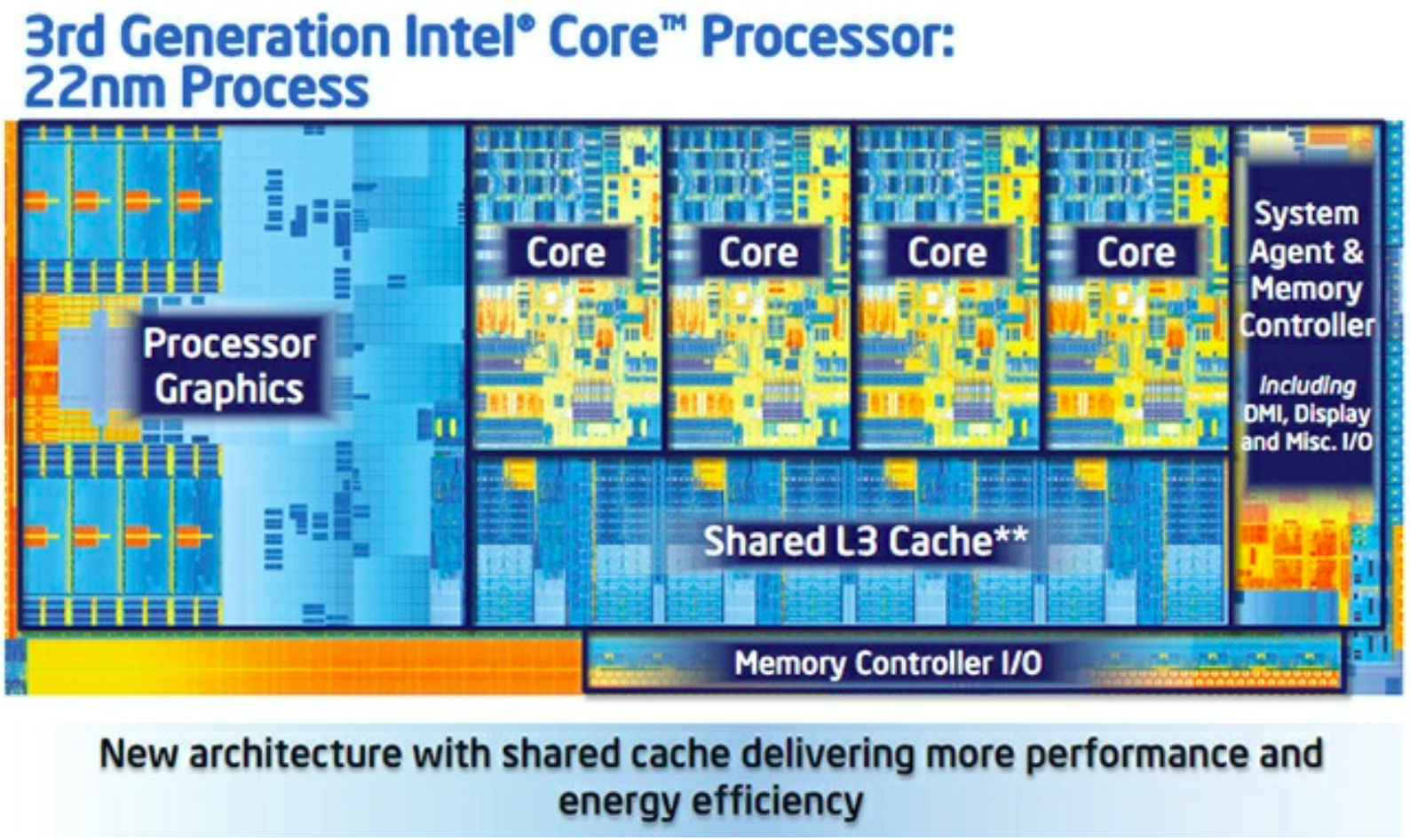
- Design appropriate algorithms
- Exploit commodity hardware

NVIDIA & AMD GPU Compute Accelerators

AMD Radeon 7970	NVIDIA GTX 680
3.79 Single Tflops	3.09 Single Tflops
947 Double Gflops	1.1 Double Tflops
2048 Stream Cores	1536 CUDA Cores

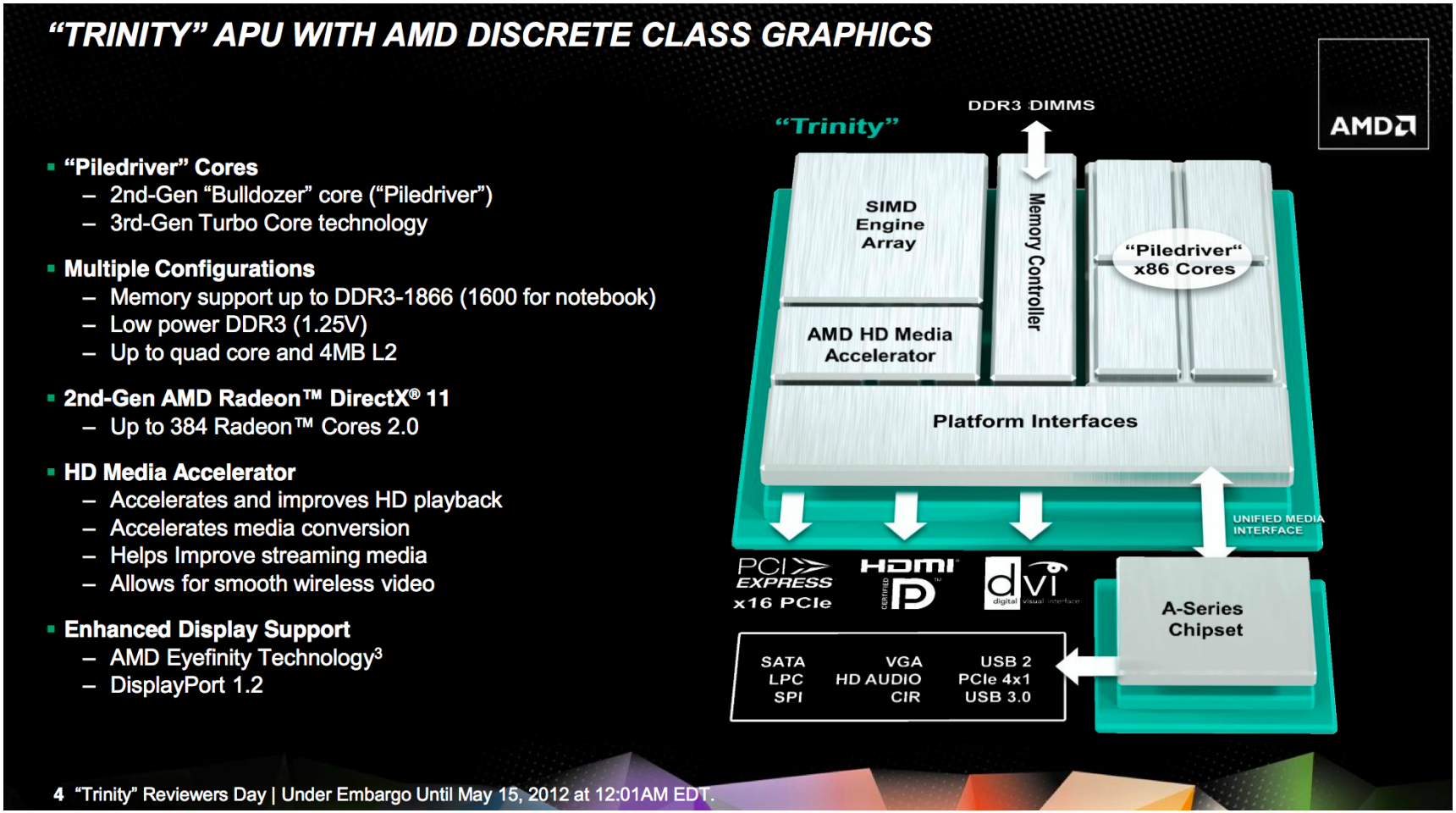
Commodity Tera-Flop Processor (peak performance)

Heterogeneous Processing



Can be programmed using OpenCL

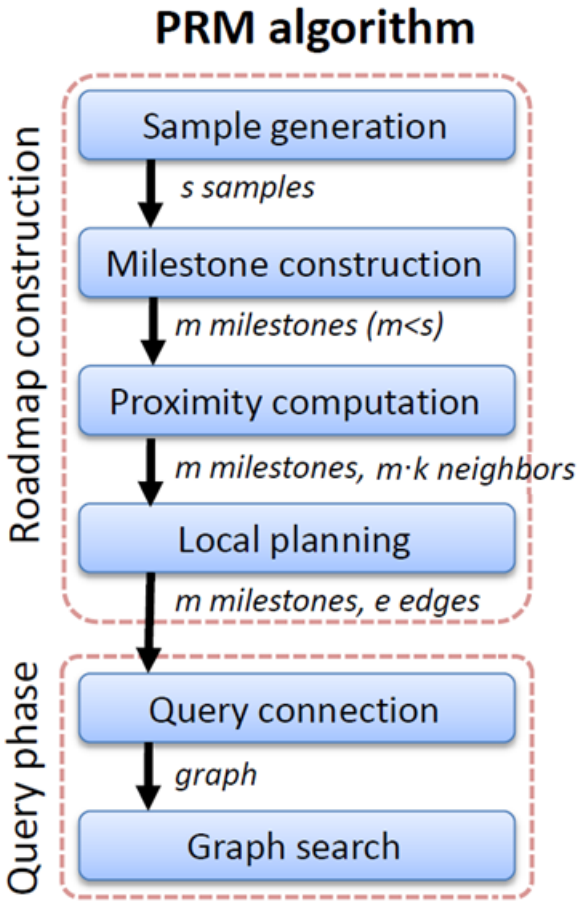
Heterogeneous Processing
AMD Trinity APU



Can be programmed using OpenCL

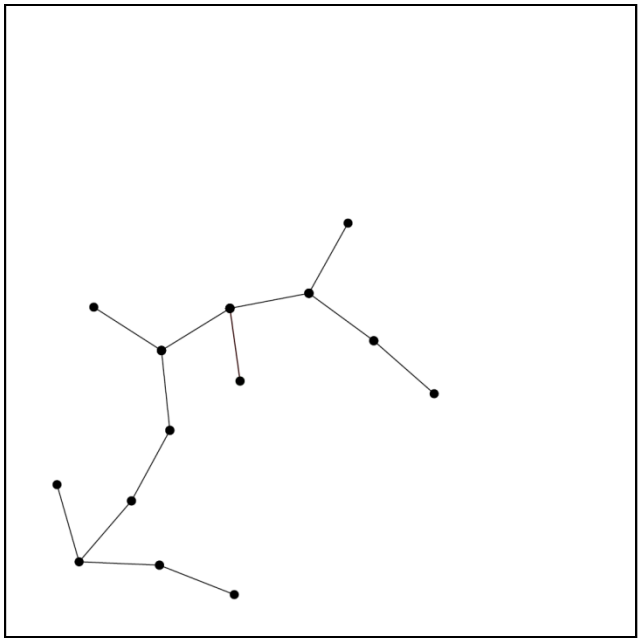
GPU-based Sampling Algorithms

- Parallel PRM algorithm on GPUs
  - G-Planner [Pan et al. 2010]
  - 10-100x speed-up from single-thread CPU algorithm
  - PRM is GPU-friendly
    - A large number of samples
    - Independent computations
    - Useful for multiple queries

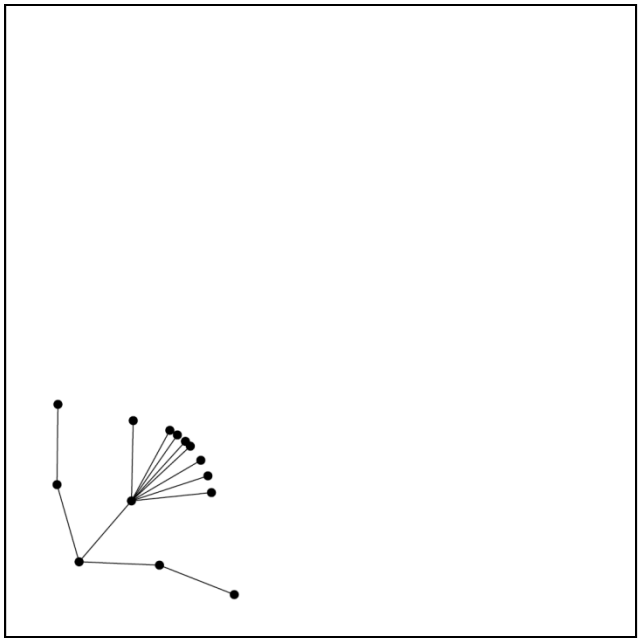


AND Parallel RRT Algorithm

- Serial RRT
  tree expansion



- AND Parallel RRT
  tree expansion



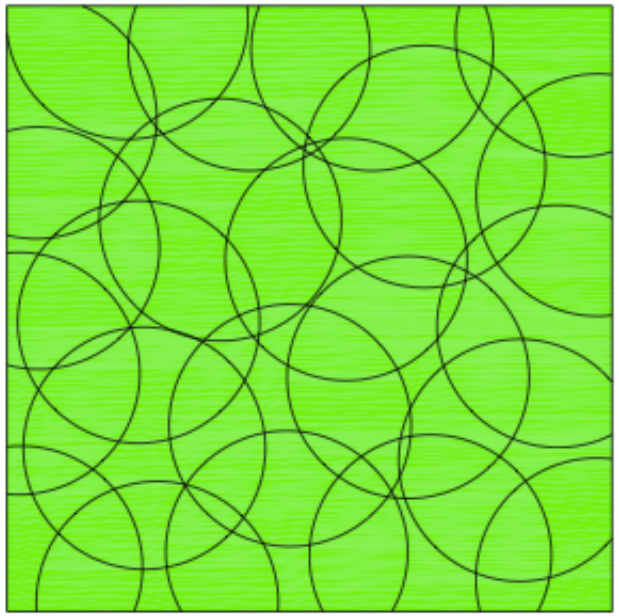
  - Generate nodes
    which are too close
  - Worse effect on GPU algorithm

Maximal Poisson-Disk Sampling

- Maximal property

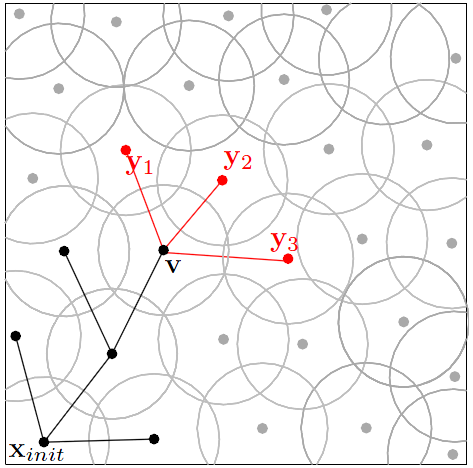
$$\forall x_i \in \mathcal{D}, \exists x_i \in X : \|x - x_i\| < r$$

  - No uncovered region in the domain by disks of radius *r*



Our Approach

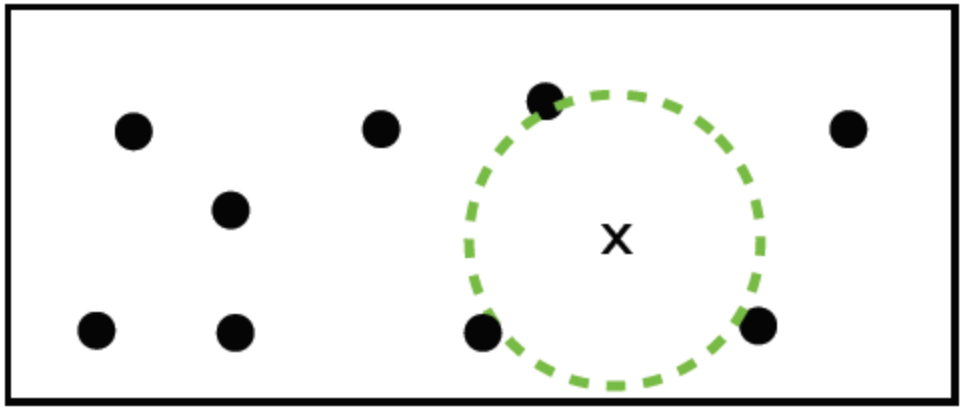
- Use maximal Poisson-disk samples in parallel RRT tree expansion
  - Empty-disk property
    - Ensure nodes are not too close
  - Maximal property
    - Ensure samples cover the entire space
  - Perform random adaptive sampling



Precomputed MPS for Planning

- Dispersion
  - The largest empty “ball” of unoccupied space

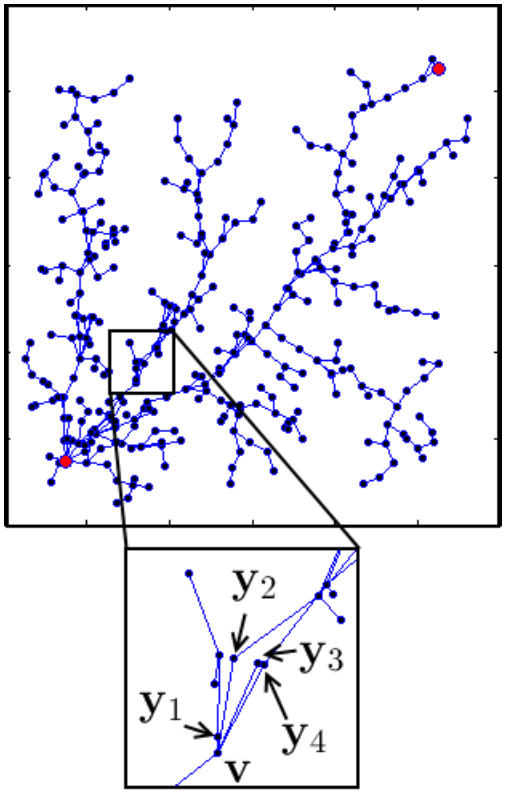
$$\delta(P, \rho) = \sup_{x \in X} \min_{p \in P} \rho(x, p)$$



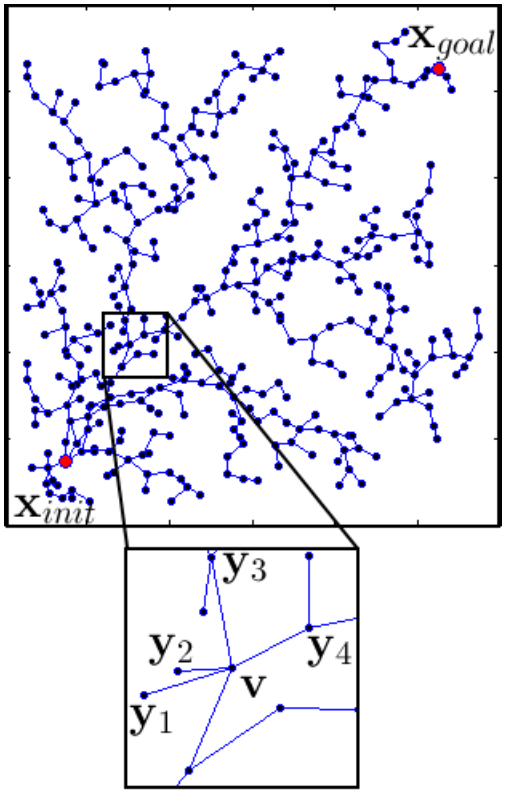
- Dispersion of MPS < r

Parallel Poisson-RRT Algorithm

- AND Parallel RRT Tree



- Poisson-RRT Tree



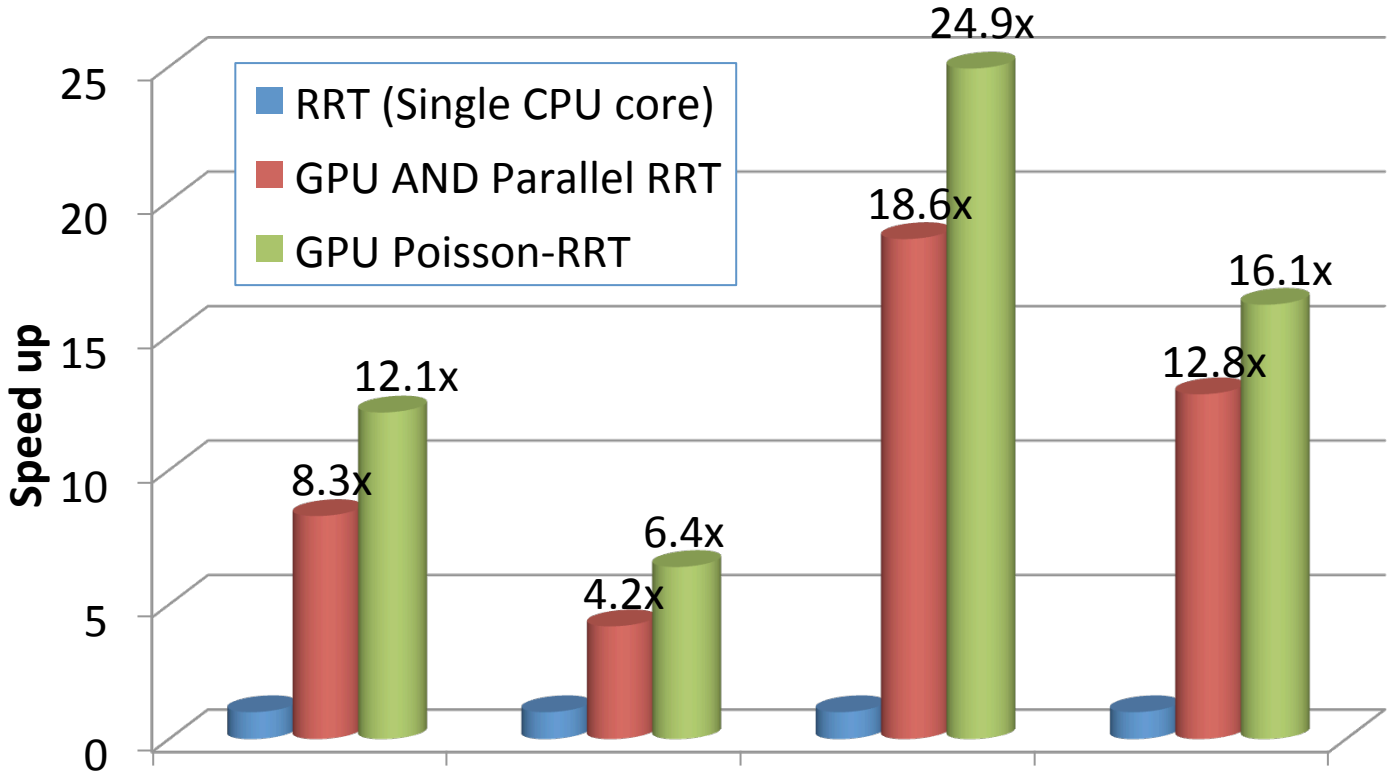
No nodes which are too close to each other

Experimental Results

- Implementation with CUDA (NVIDIA GPUs)
- Integrated in OMPL and ROS simulator
  - 3D(6-DOFs) OMPL benchmarks
  - HRP-4 robot planning(23-DOFs)
- System
  - CPU: Intel Sandy Bridge i7-2600 (Single thread)
  - GPU: NVIDIA Geforce GTX580

Experimental Results

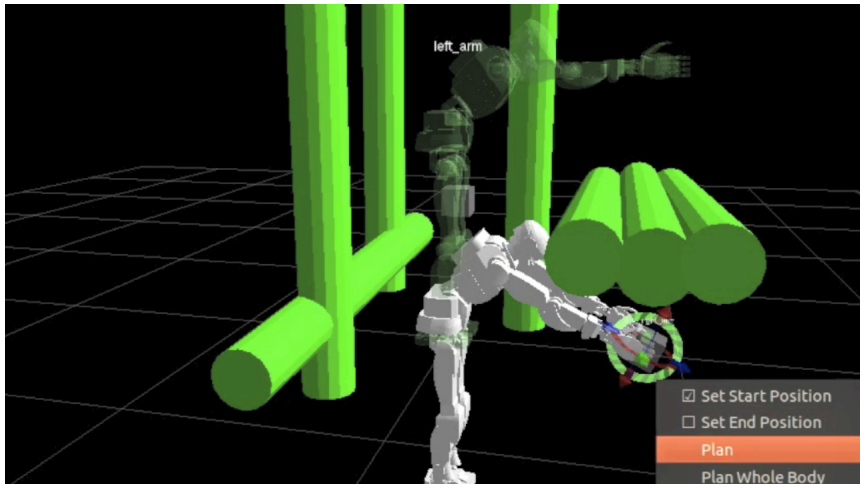
- Planning Time for OMPL Benchmarks

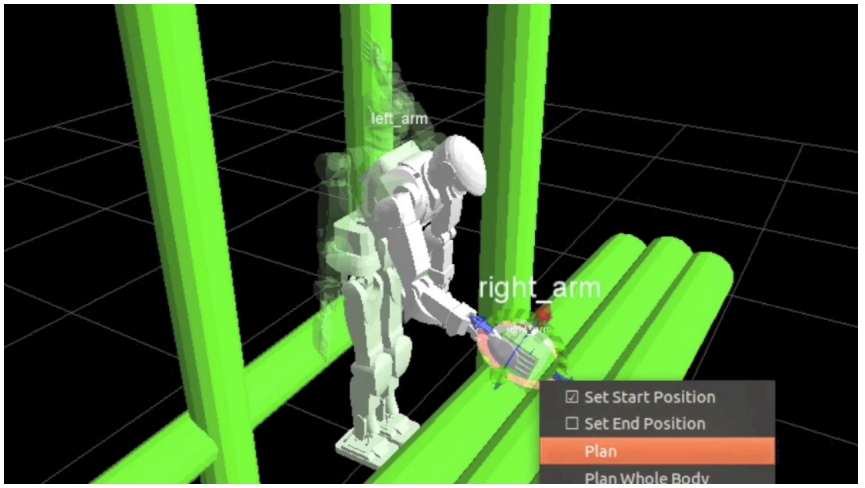


Absolute planning time for GPU Poisson-RRT

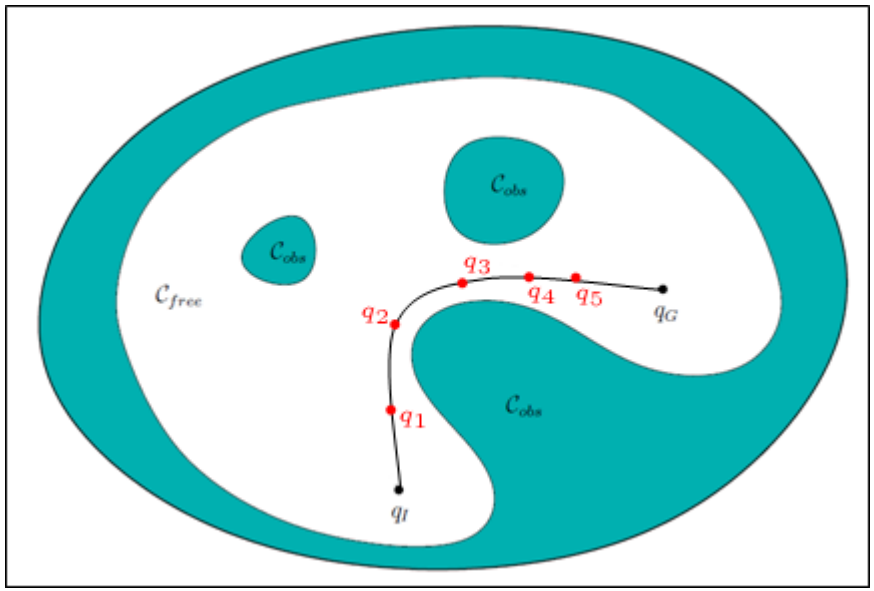
Experimental Results

- HRP-4 robot planning (23 DOFs): Real-time RRT Planner





Optimization-based Planning Algorithm

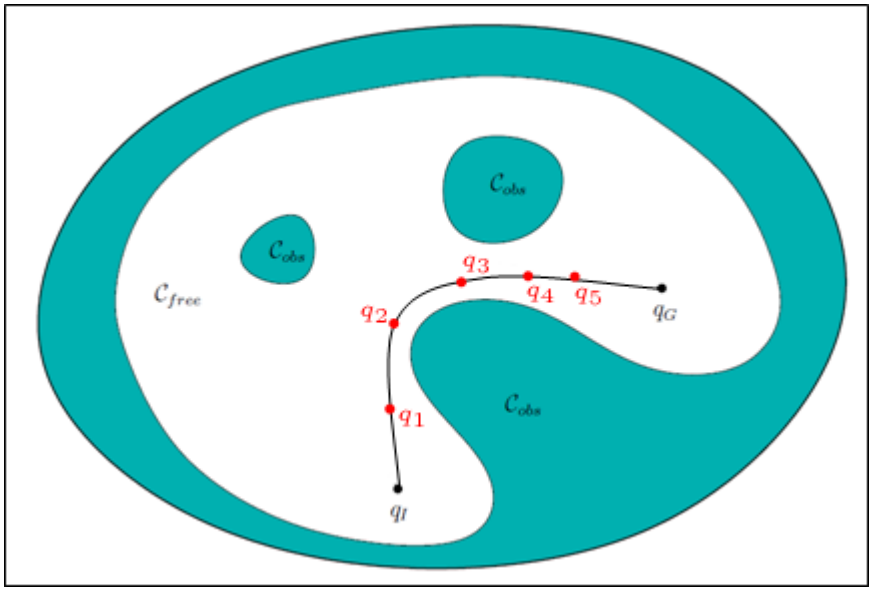


- Start from an initial trajectory
  - The trajectory is discretized into waypoints $q_I, q_1, \cdots, q_N, q_G$, based on a uniform time interval

Optimization-based Planning Algorithm

- Optimization objective function

$$\min_{\mathbf{q}_1,\ldots,\mathbf{q}_N} \sum_{i=1}^{N} \left(c(\mathbf{q}_i) + \|\mathbf{q}_{i-1} - 2\mathbf{q}_i + \mathbf{q}_{i+1}\| \right)$$



$c(\mathbf{q}_i)$: Costs for $\mathbf{q}_i$ (**collisions**, …)

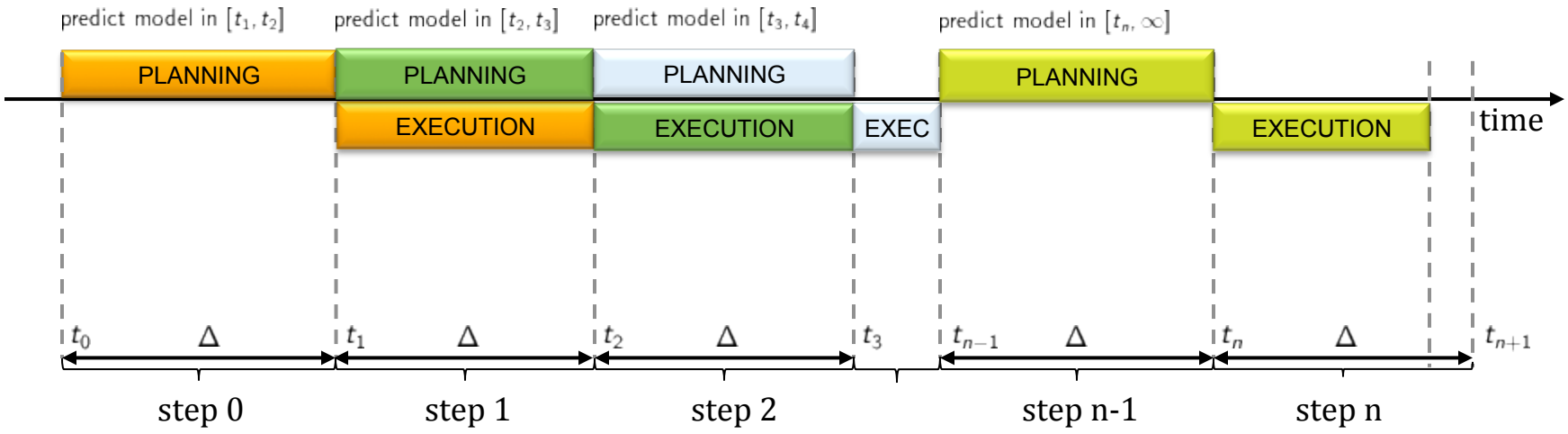
- Static and dynamic obstacles

$\sum_{i=1}^{N} \|\mathbf{q}_{i-1} - 2\mathbf{q}_i + \mathbf{q}_{i+1}\|$: Costs for **smoothness**

- Approximates the squared acceleration

Planning in Dynamic Environments

- Incremental Trajectory Optimization
  - Compute partial plan for the next execution step
  - Improve the trajectory during execution
  - Use the latest sensor information



Parallel Trajectory Optimization

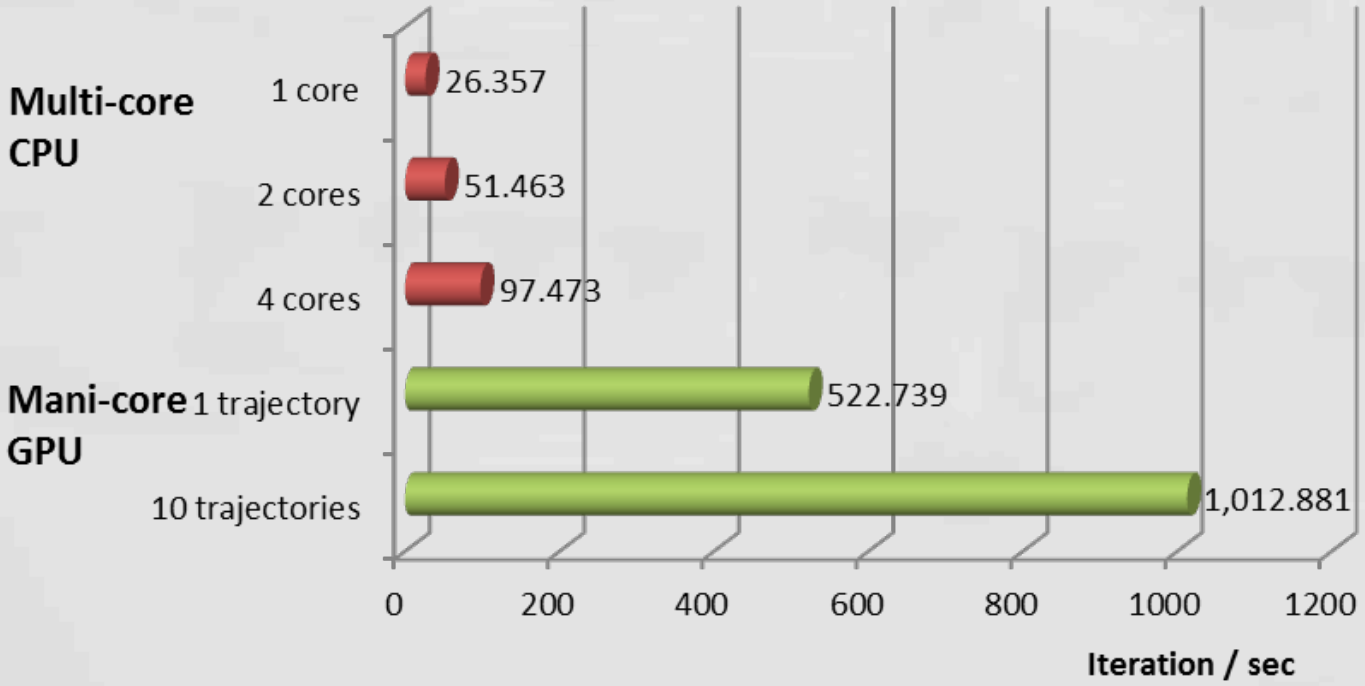
- Parallel optimization of multiple trajectories
  - Use Multiple threads
    - Start from different initial trajectories
    - Trajectories are generated by quasi-random sampling
  - Exploits the multiple CPU cores (multi-cores) or GPU-based cores (many-cores)

Parallel Trajectory Optimization

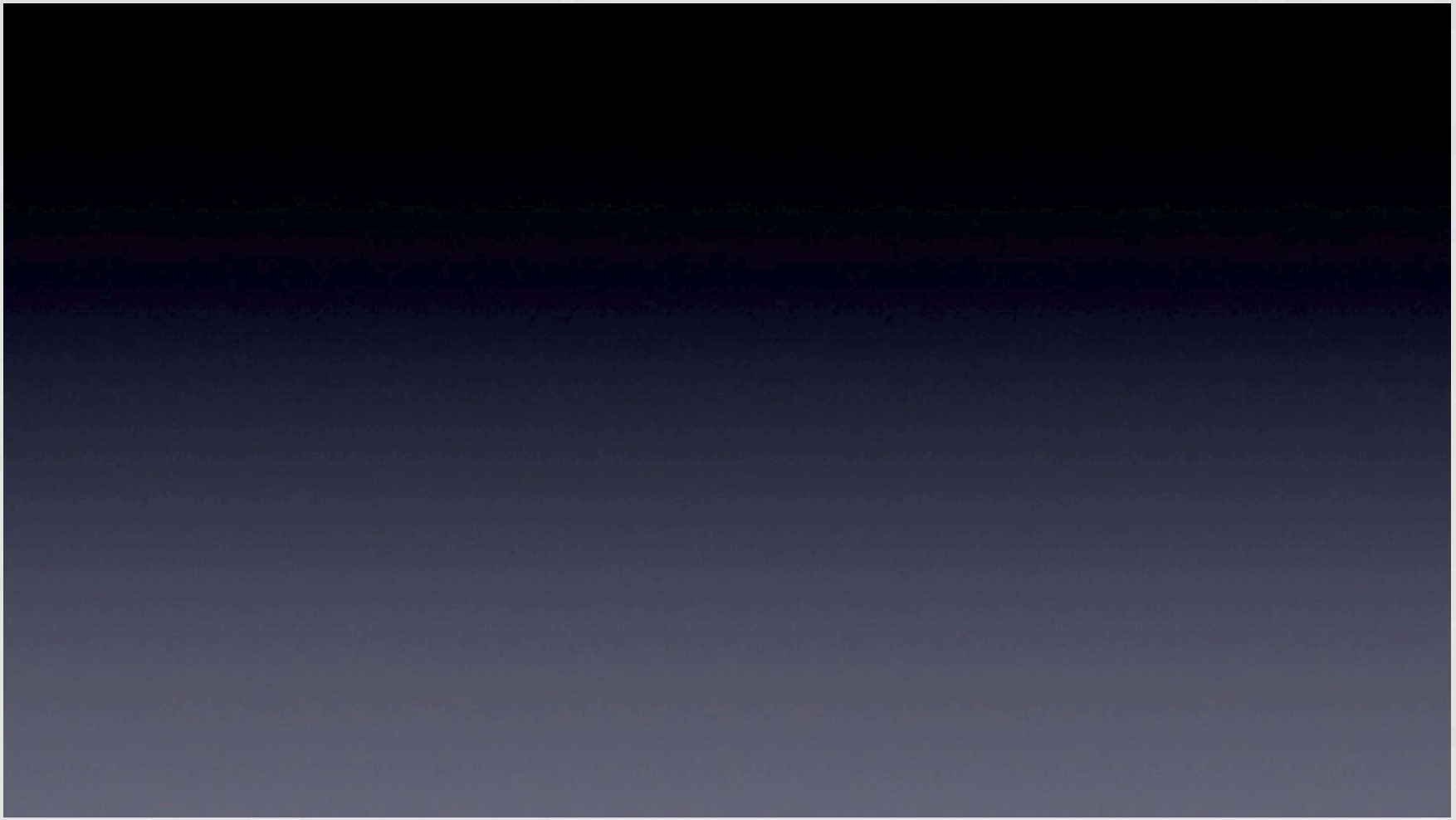
- Parallelization improves the performance
  - Reduce the iteration time of the single optimization
    - Parallel collision checking and constraint handling
  - Parallel optimization of multiple trajectories reduces the time to compute the first collision-free solution

Parallel Trajectory Optimization

Performance improvement with number of cores



Human-Like Environment: Real-Time Planning



Ackowledgements

- Army Research Office
- National Science Foundation
- Intel
- Willow Garage