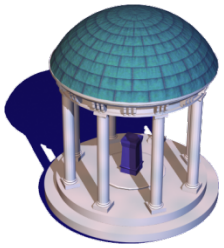
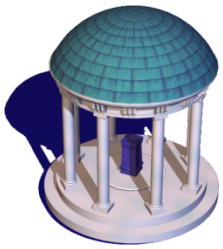


Real-Time Motion Planning and Handling Model Uncertainty

Dinesh Manocha
Department of Computer Science
University of North Carolina, Chapel Hill
http://gamma.cs.unc.edu

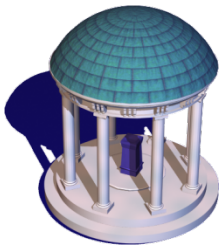


Collaborators

- Sachin Chitta (Willow Garage)
- Christian Lauterbach
- Jia Pan

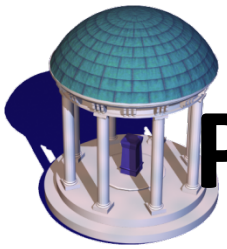
Motion Planning: Applications

- Manufacturing:
  - Design of part feeders
- Design for manufacturing and servicing
- Design of pipe layouts and cable harnesses
- Autonomous mobile robots planetary exploration, surveillance, military scouting
- Graphic animation of "digital actors" for video games, movies, and webpages
- Virtual walkthru
- Medical surgery planning
- Generation of plausible molecule motions, e.g., docking and folding motions
- Building code verification



Sample-based Planners

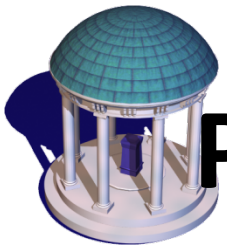
- Collision checking is one of the major bottlenecks
- Can take more than 90% of total running time



Prior work on collision checking

- Fast algorithms for convex polytopes
- Bounding volume hierarchies for general polygonal models
- Deformable models
- Multiple systems

I-Collide, RAPID, PQP, DEEP, SWIFT, SWIFT++, DeformCD, Self-CCD,…..

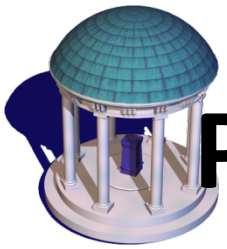


Prior work on collision checking

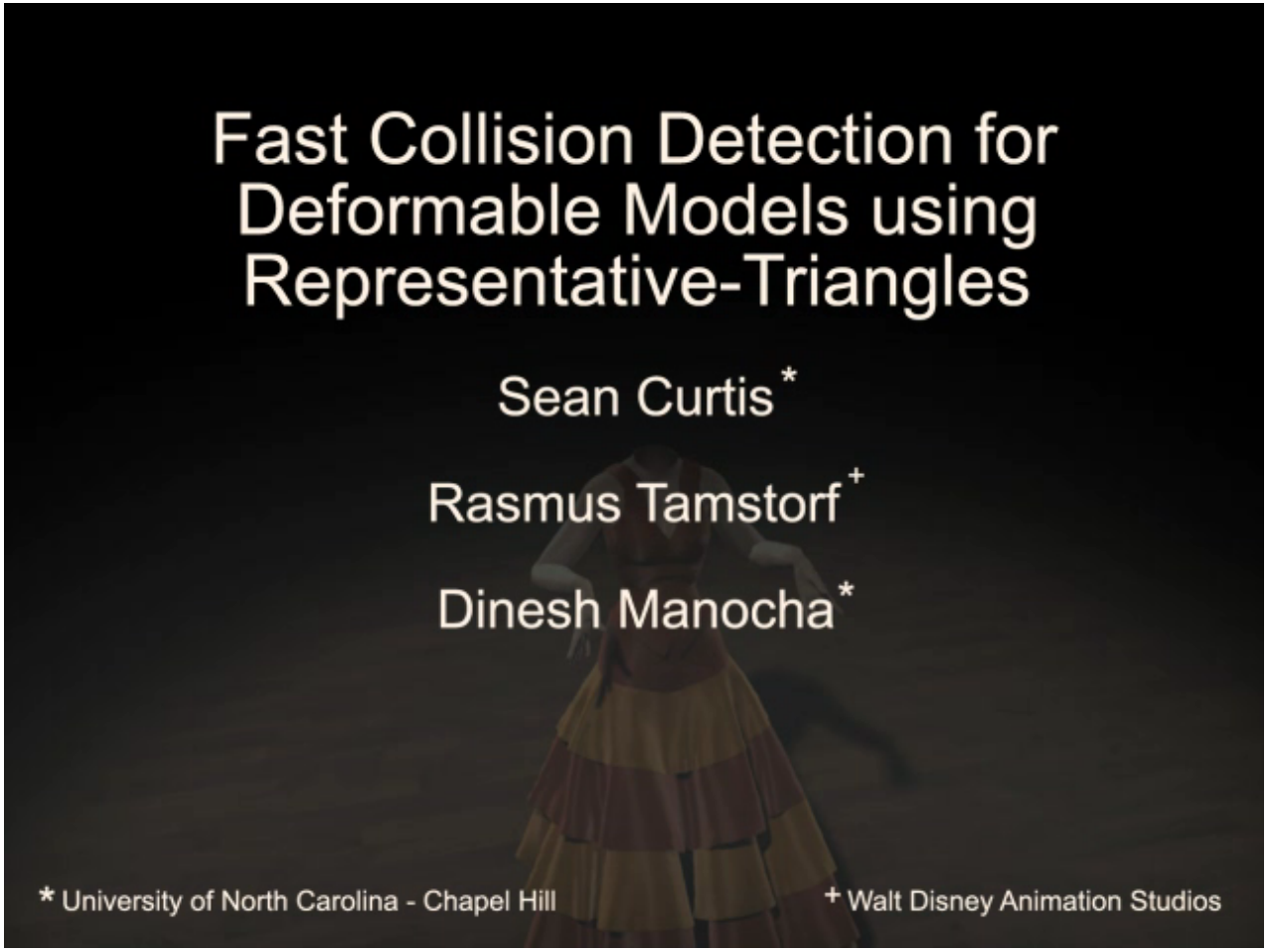
- Fast algorithms for convex polytopes
- Bounding volume hierarchies for general polygonal models
- Deformable models
- Multiple systems

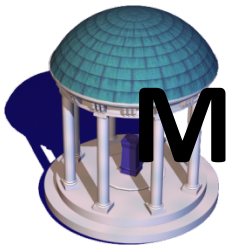
I-Collide, RAPID, PQP, SWIFT, SWIFT++, DeformCD, Self-CCD,…..

But these systems assume exact model representation using triangulated models



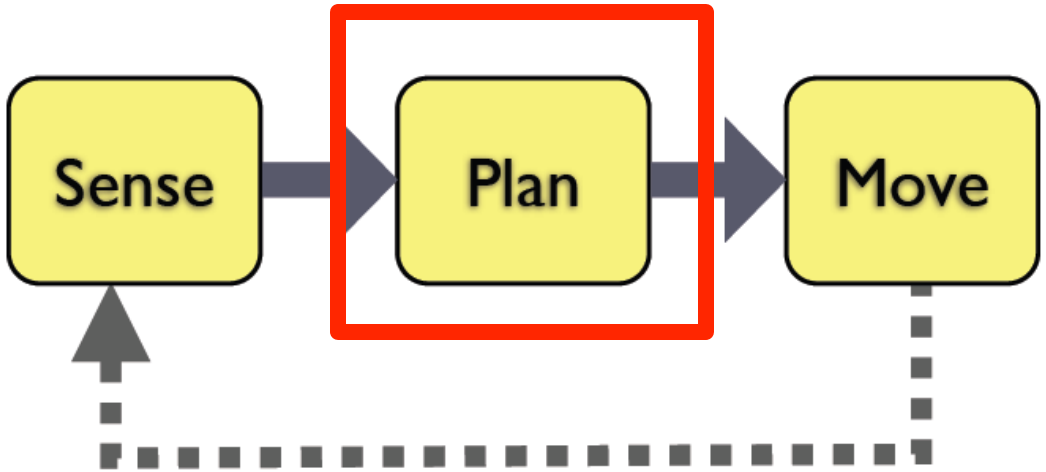
Prior work on collision checking





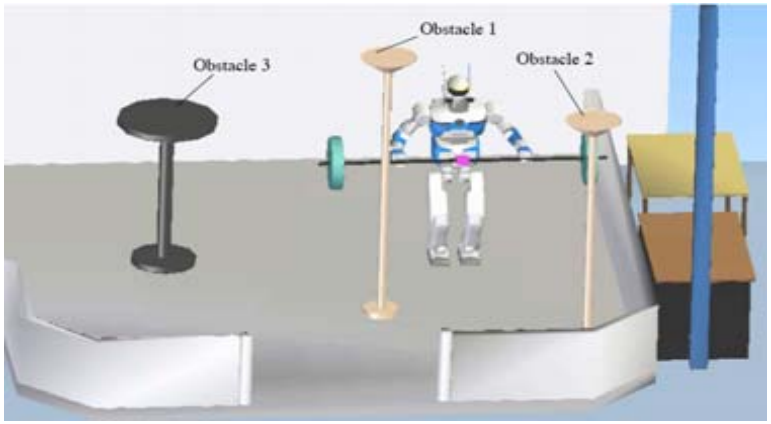
Motion Planning for Physical Robots

- Planning a complex 3D task requires complex 3D models
  - Task level planning vs. motion planning

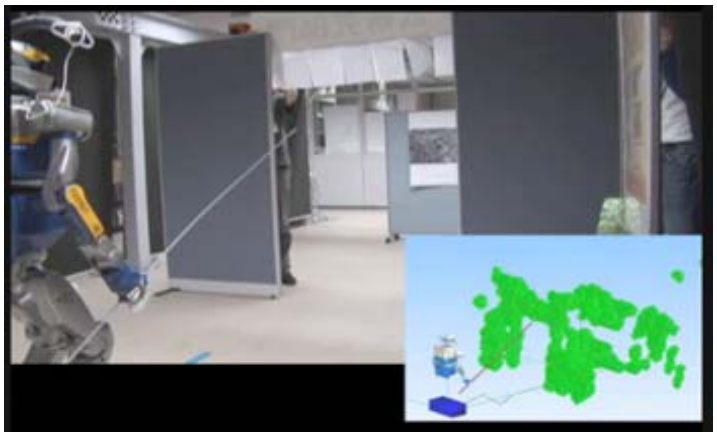


- Task execution needs real-time feedback to follow dynamic/ uncertainty environment
- How to transform arbitrary tasks to a sequence of motion planning problems? Link with Perception?

Motion Planning for Physical Robots



The real world is not so nice as virtual reality!



The perceived world!

- Robots use sensors such as cameras, Lidar, tactile, which provide only partial information about the physical world

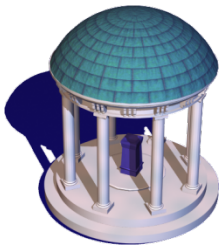
- Sensor and actuator error; real-time data processing [Laumond 2010]

Motion Planning for Physical Robots

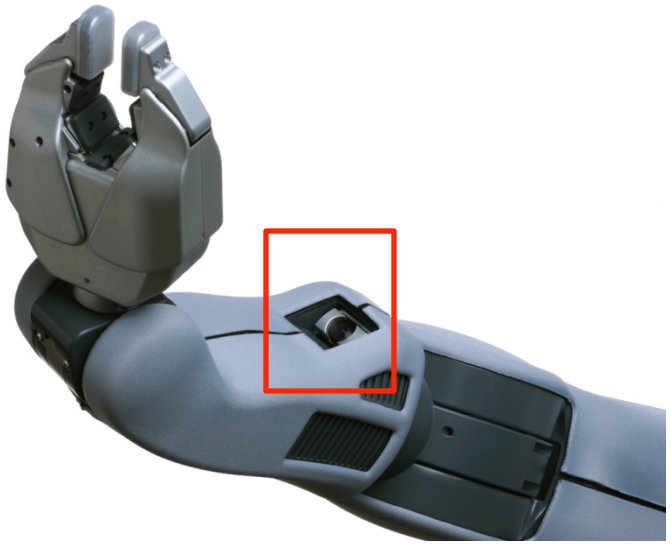
- Collision checking on noisy point cloud data
- Real-time high DOF planning using graphics hardware

Motion Planning for Physical Robots

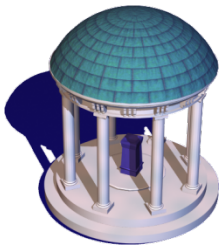
- Collision checking on noisy point cloud data
- Real-time high DOF planning using graphics hardware



Robot Sensors: Data Collection



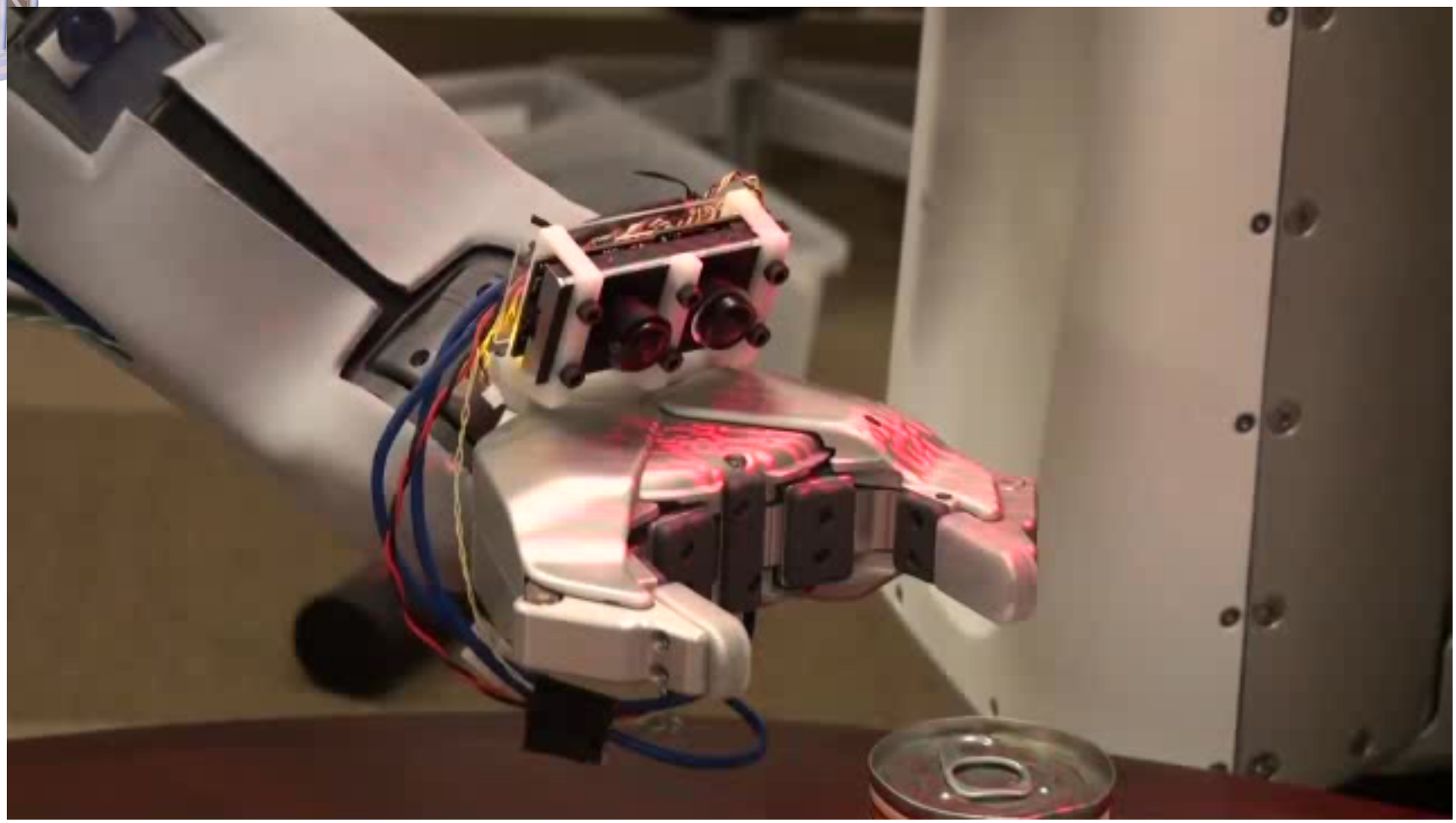
Cameras



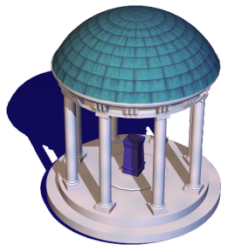
Robot Sensors: Data Collection

Laser Scanners

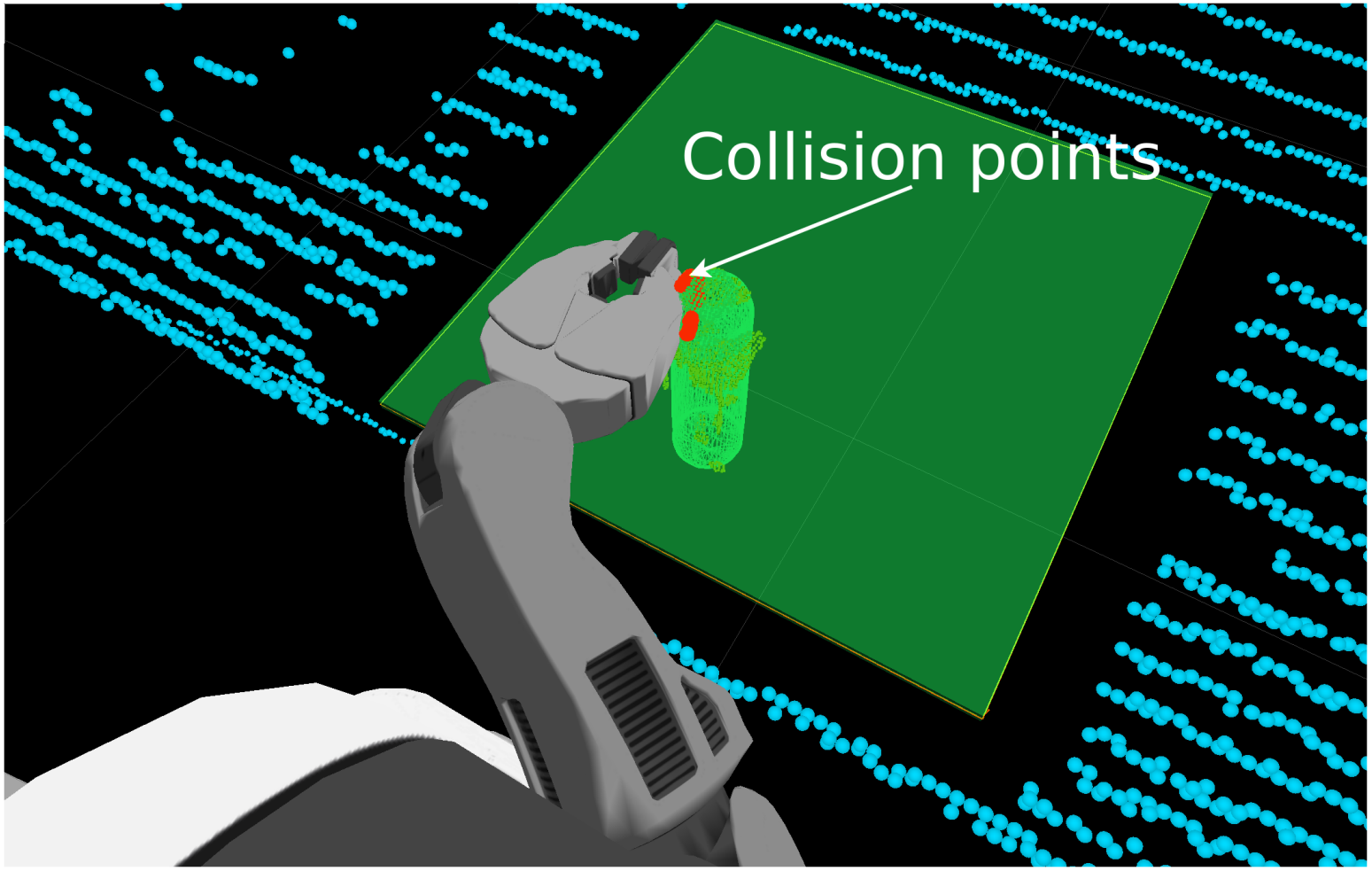
Demonstration of Point Cloud Data

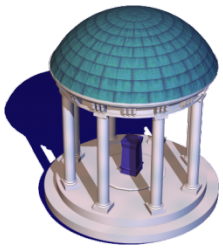


Integration with PR2 Sensors (Willow Garage)



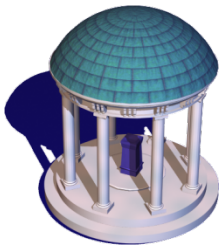
Reconstructed Point Clouds





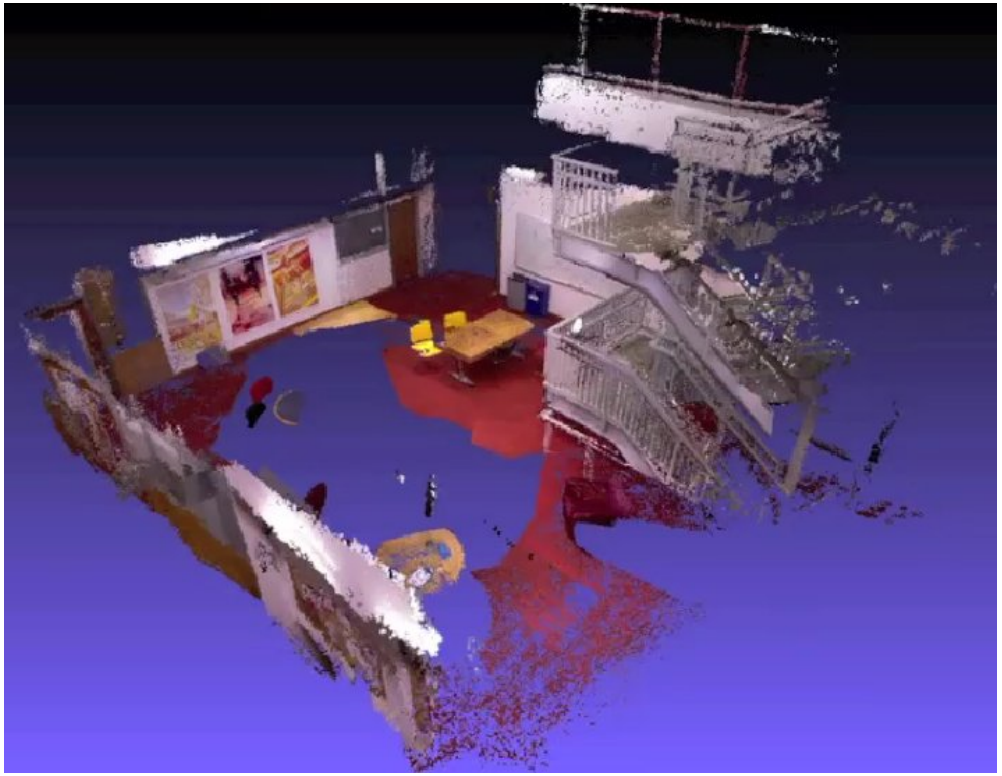
Kinect Sensors

http://graphics.stanford.edu/~mdfisher/Kinect.html

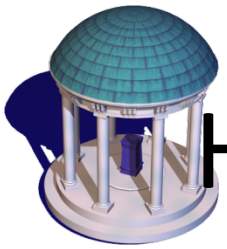


Kinect Reconstruct Result

http://www.cs.washington.edu/ai/Mobile_Robotics/projects/rgbd-3d-mapping/

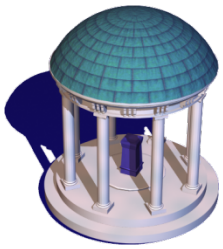


http://groups.csail.mit.edu/rrg/index.php?n=Main.VisualOdometryForGPS-DeniedFlight

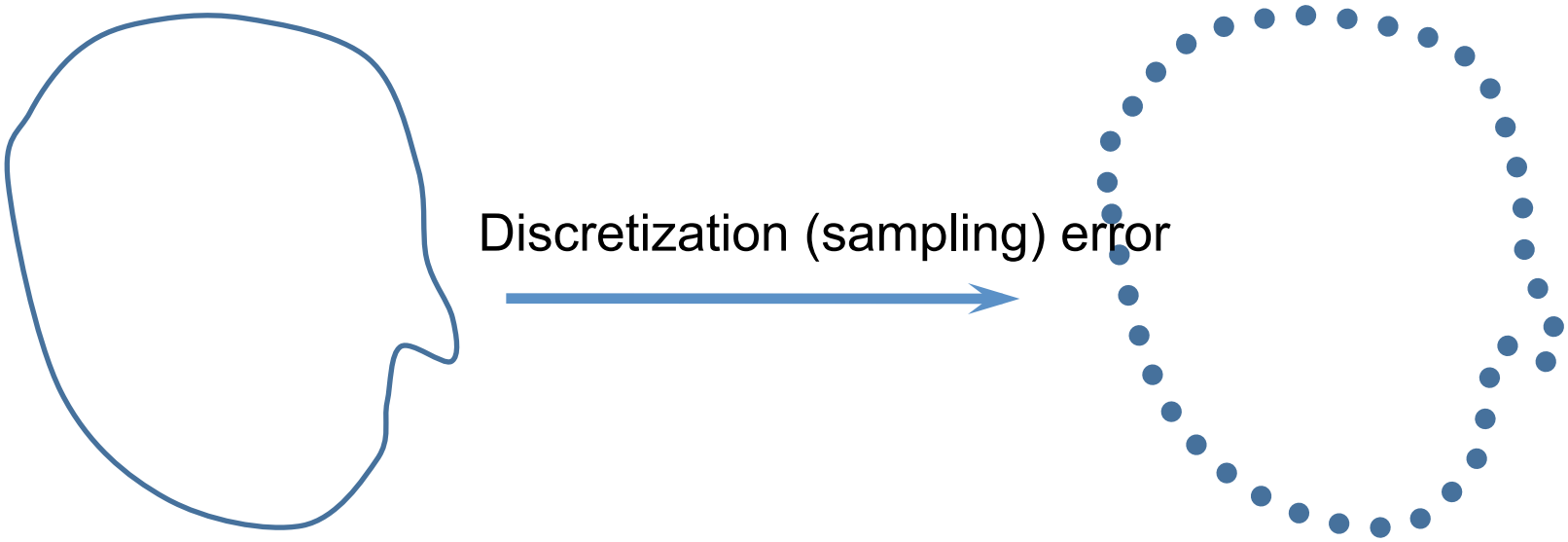


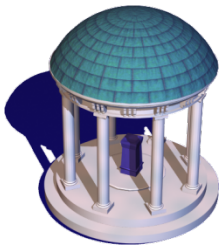
Handling Noisy Point Cloud Data

- Planning, navigation and grasping
- Scene reasoning
- Noisy data
- Real-time processing

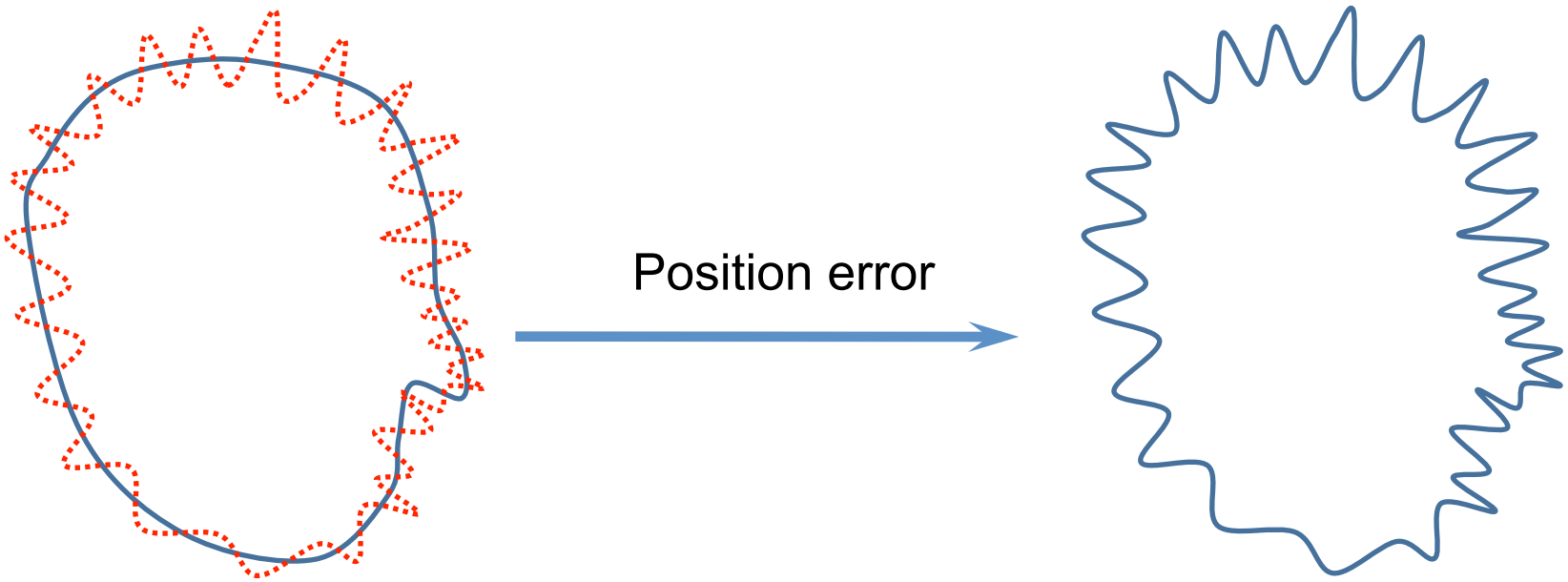


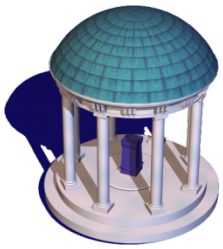
Errors in Point Clouds



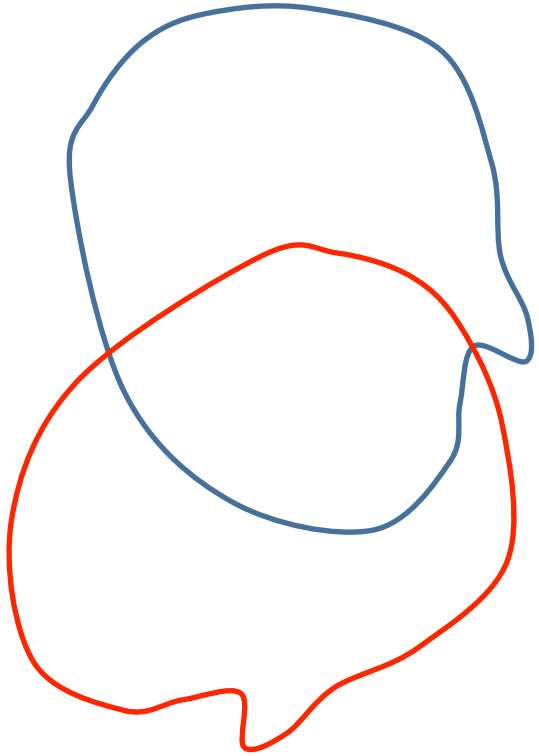


Errors in Point Clouds

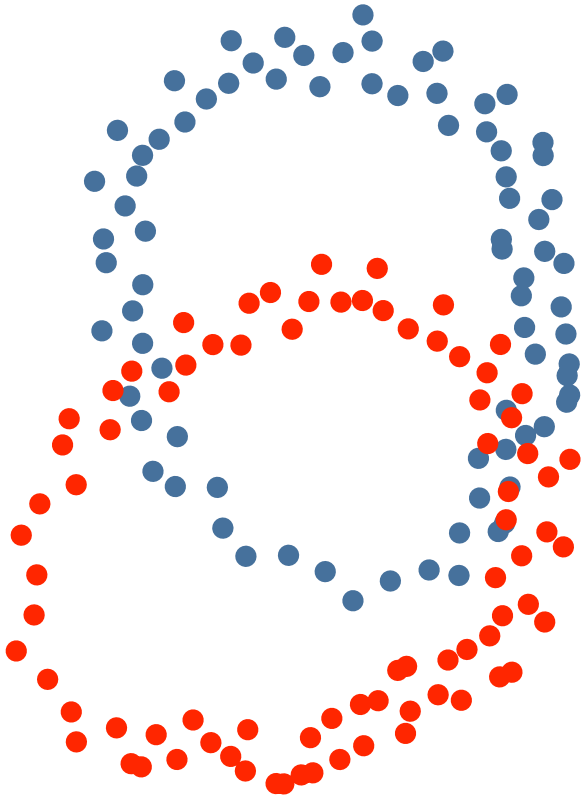




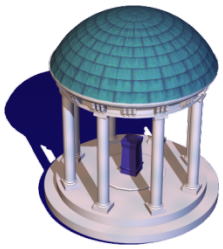
Point Cloud Collision Detection



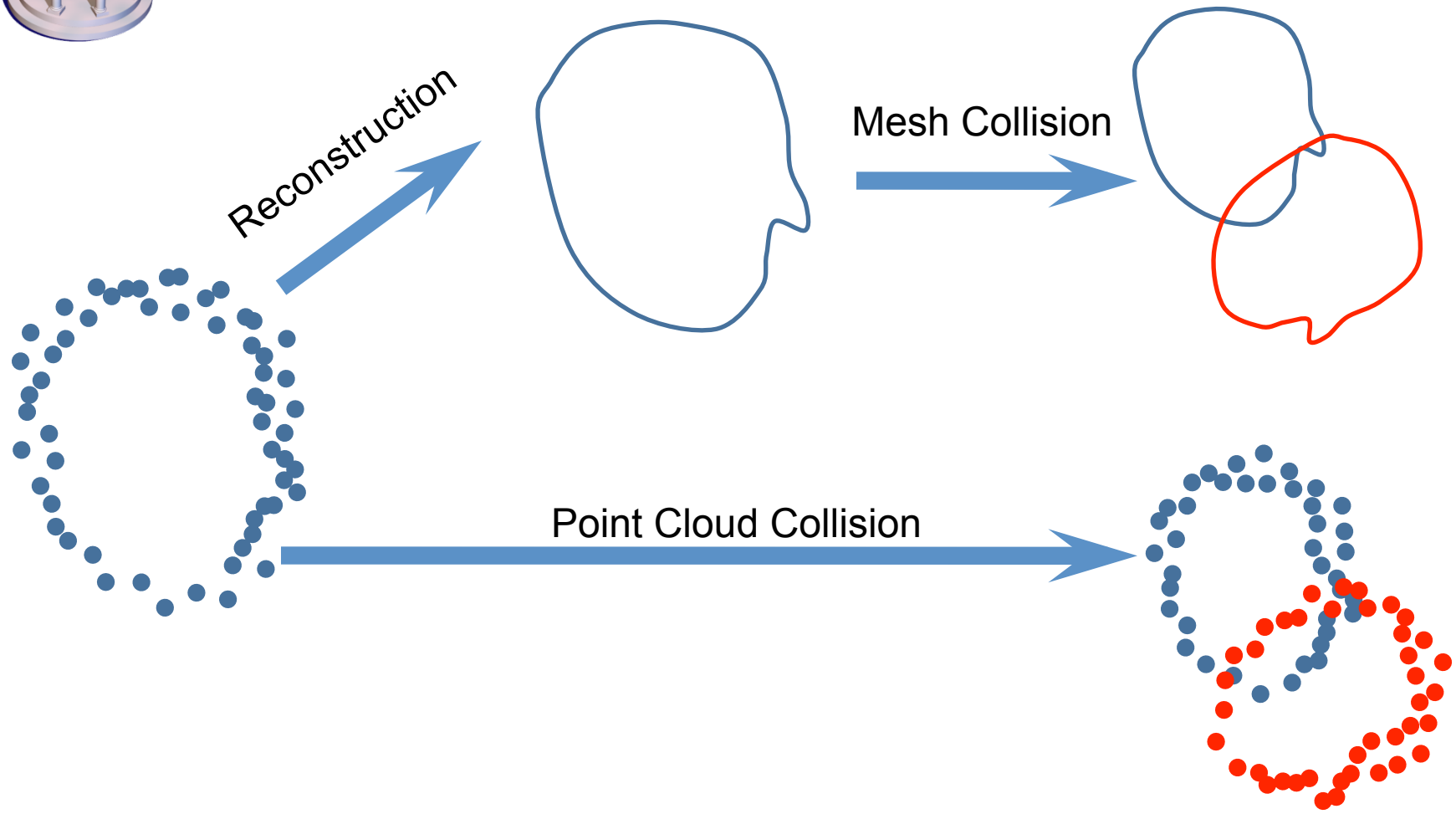
In-collision

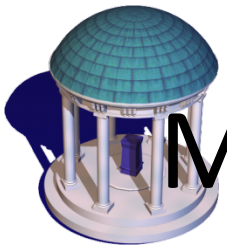


In-collision ?

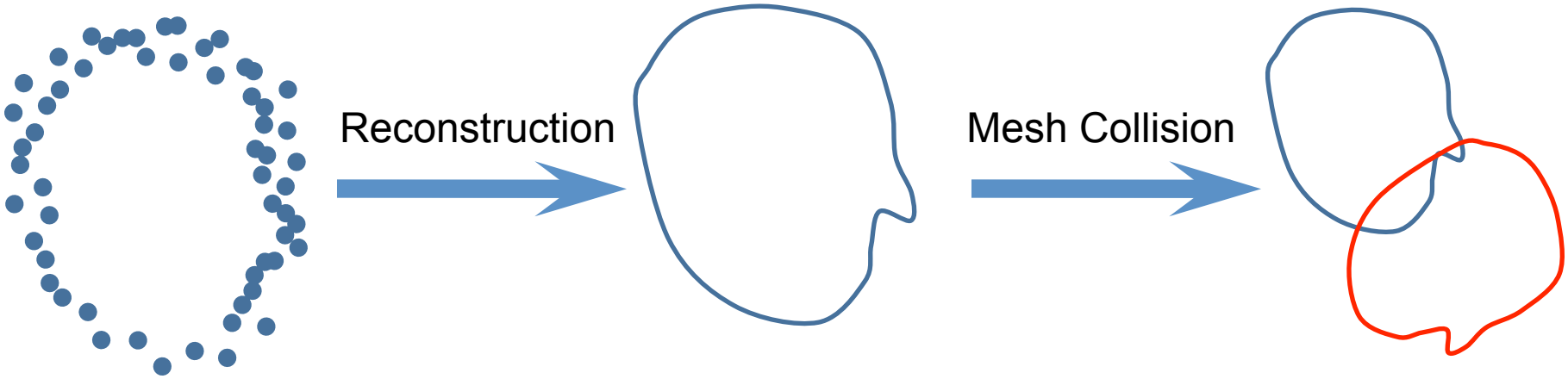


Handling Point Cloud Collision: Two Methods



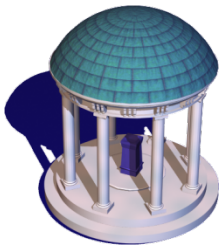


Mesh Reconstruction => Collision



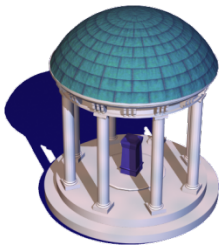
Reconstruction is more difficult then collision detection

Solve an easier problem by conquering a more difficult one?



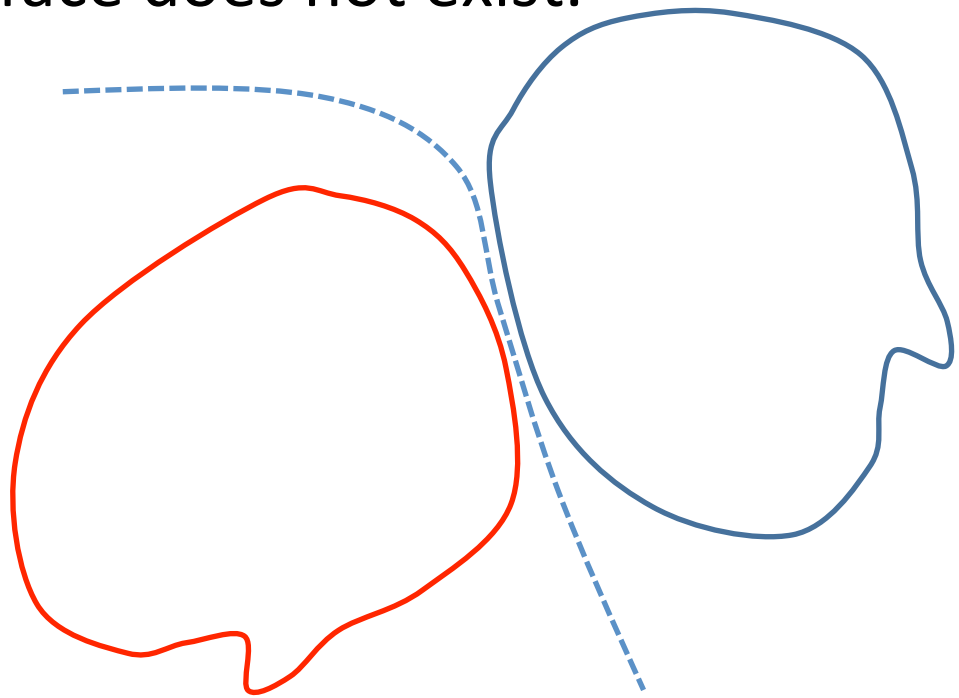
Mesh Reconstruction => Collision

- Reconstruction process is not robust, and is sensitive to noise and high order features
- Reconstruction process is slow (few seconds)
- Error in reconstructed result can be amplified by subsequent collision checking
- The final result is YES/NO answer, which is sensitive to noise.



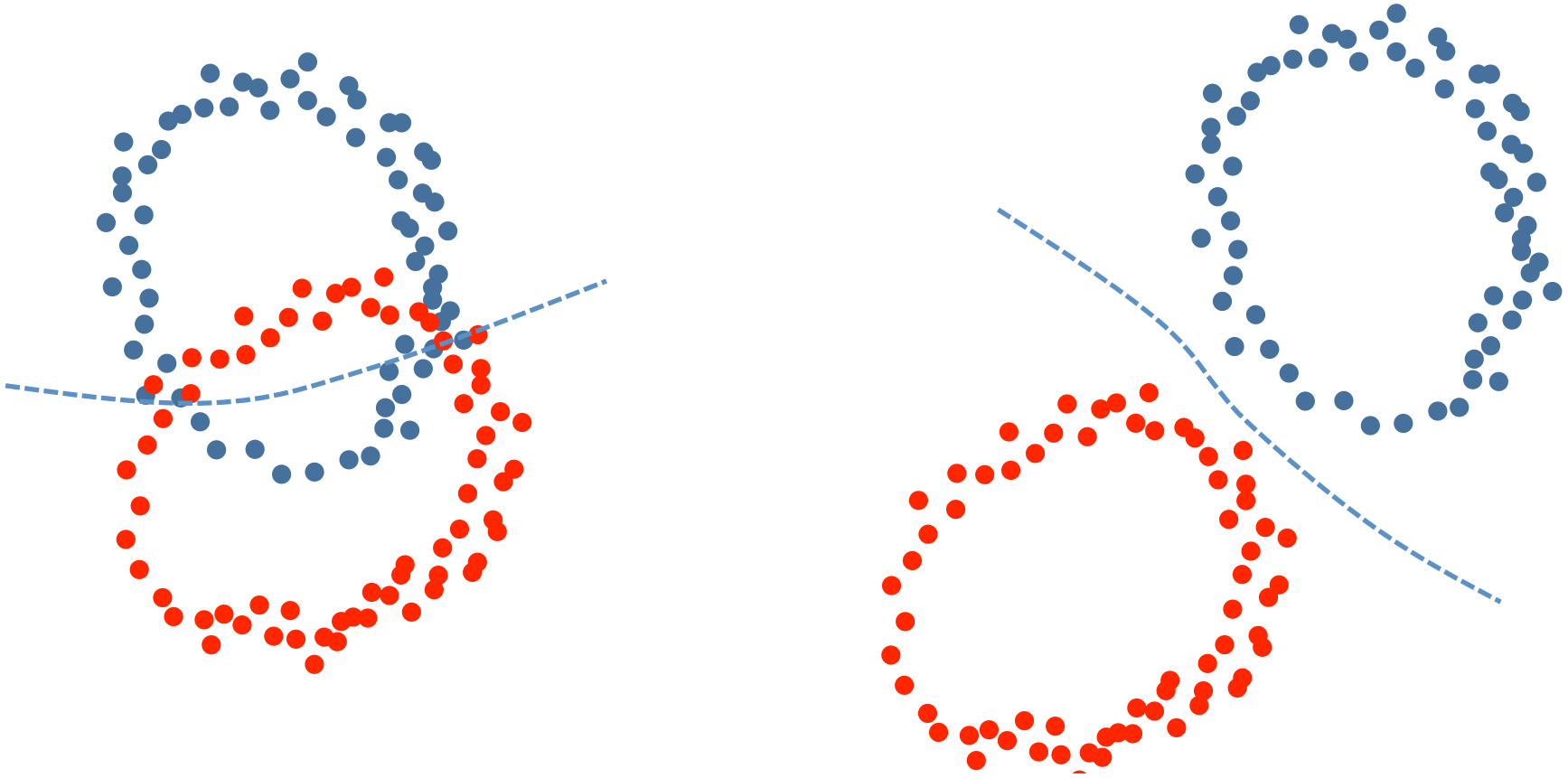
Our Solution

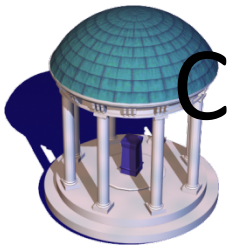
- Return to the basic definition of collision-free
  - Two objects are collision-free if they are separable by a continuous surface and is in-collision when such surface does not exist.



Classification-based Collision Detection

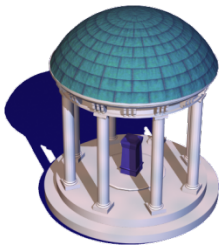
- Find a separating surface that separates two points clouds as much as possible



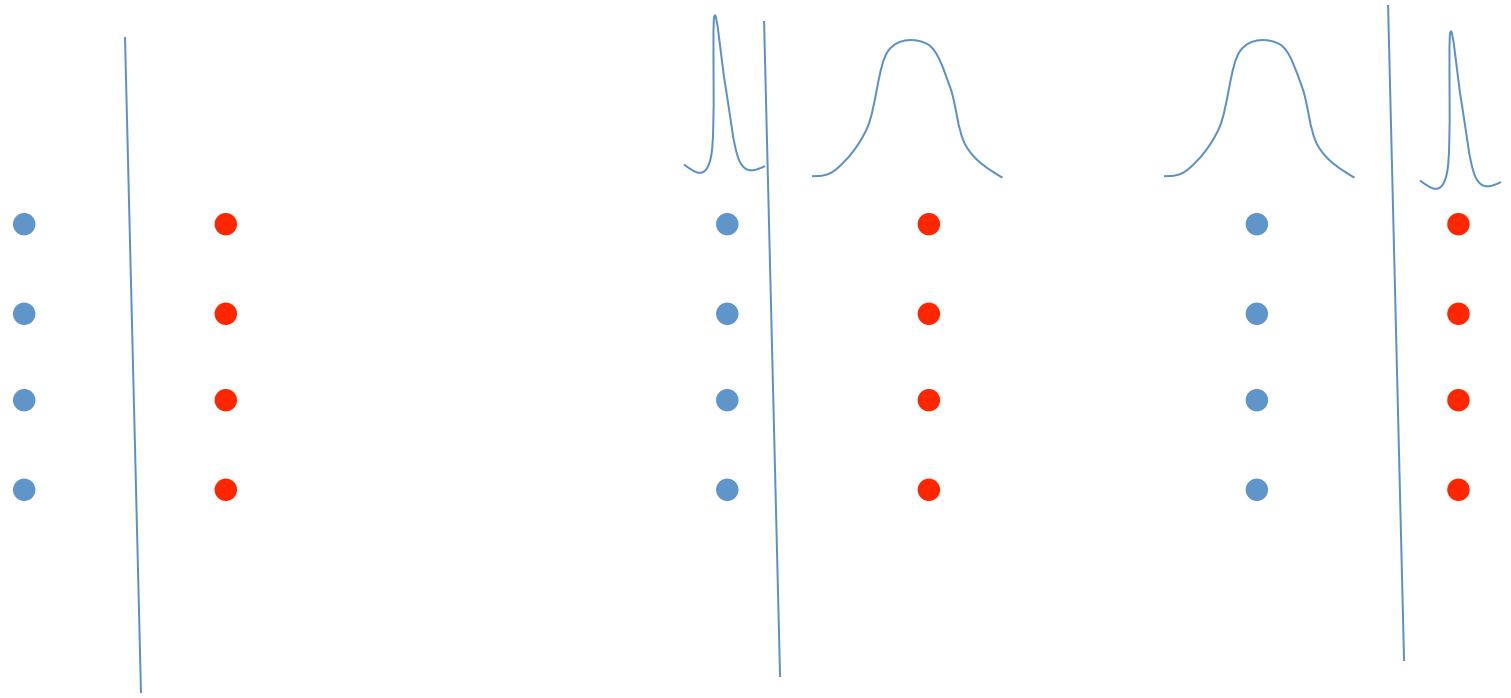


Collision Detection based on Robust Classification

- We find the optimal (i.e. minimize the separating error) separating surface using a SVM-like algorithm
- Use supervised machine learning methods for geometric classification
- Different from standard SVM: each training data point has noise – corresponds to robust classification in machine learning

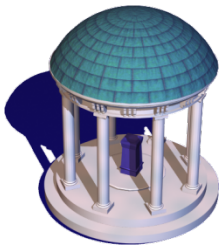


Robust Classification



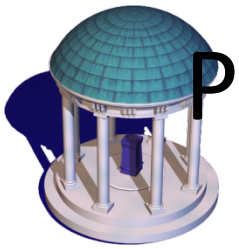
Standard SVM

Robust Classification: aware of noise



Per-point Collision Probability

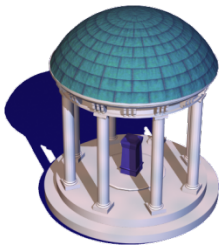
- Collision probability: the probability that one point is on the wrong side of separating surface.
- Robust classification computes collision probability for each single point sample



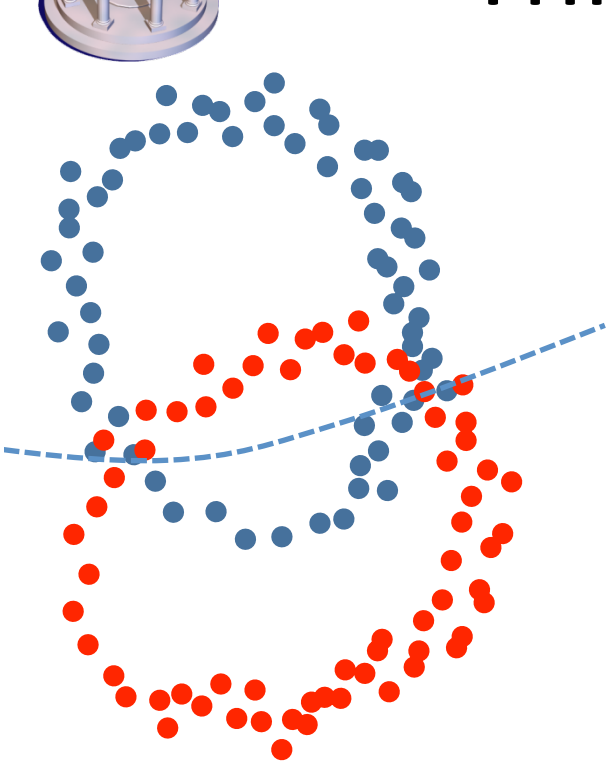
Probabilistic Collision between Two Objects

- For each object
  - Cluster the points and only keep one point in each cluster: compute collision probability for independent points
  - Filter out points whose collision probability is small: compute points with reliable collision probability (i.e. collision result will not change much when noise level changes).
  - Overall object collision probability

$$1 - \prod_{i=1}^{m} [1 - \mathbb{P}(\tilde{\mathbf{x}}_i)]$$



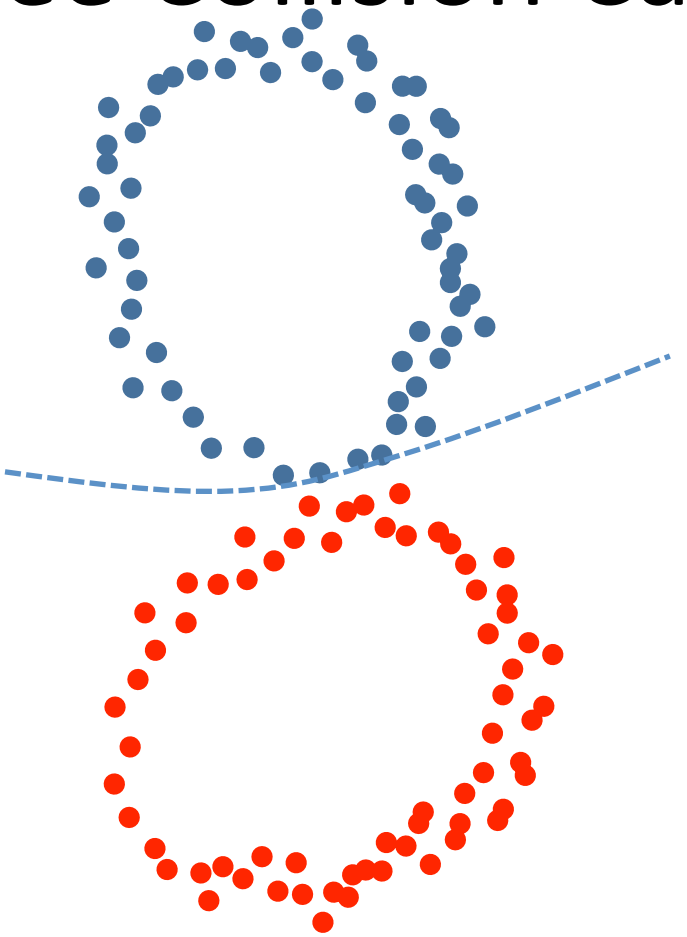
Three Collision Cases



Deep collision

Collision probability
near 1

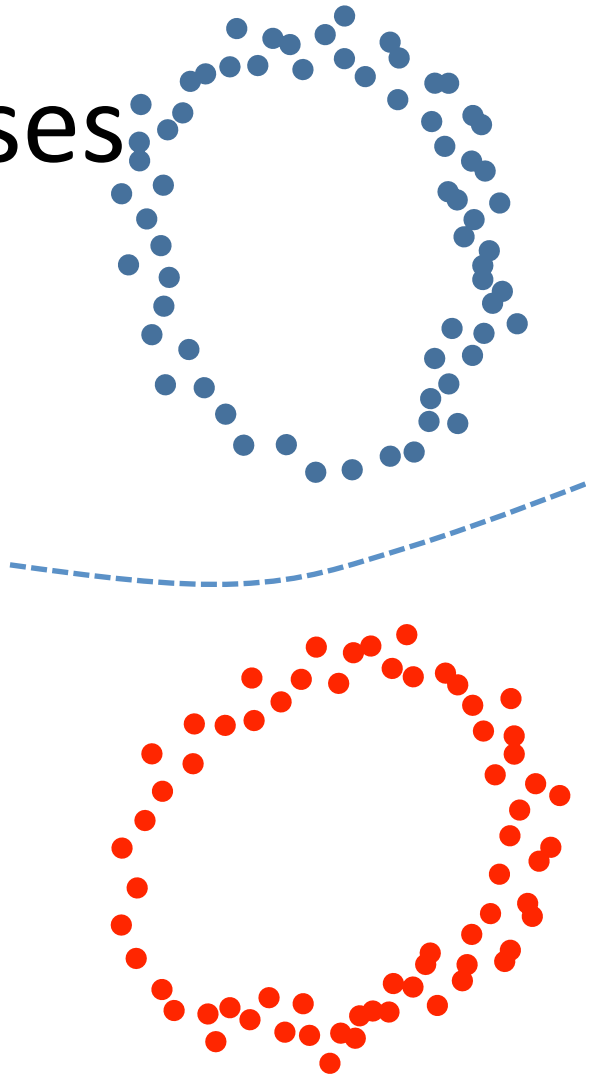
Easy



In-contact

Collision probability
near 0.5

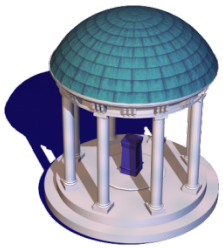
Difficult: small noise will
bring reverse of yes/no answer



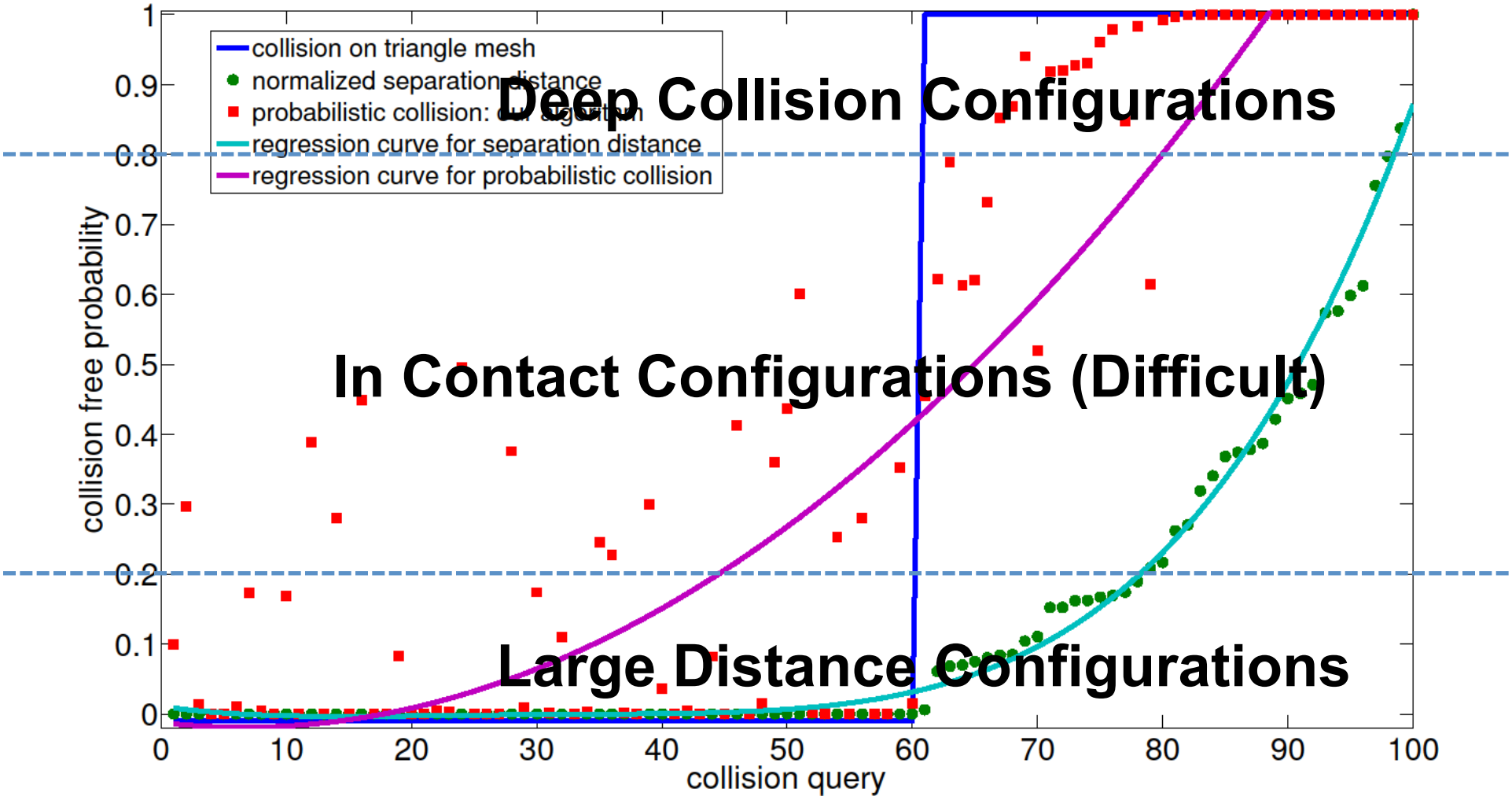
Large distance

Collision probability
near 0

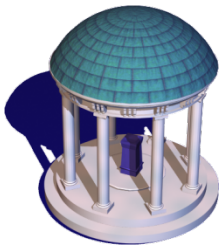
Easy



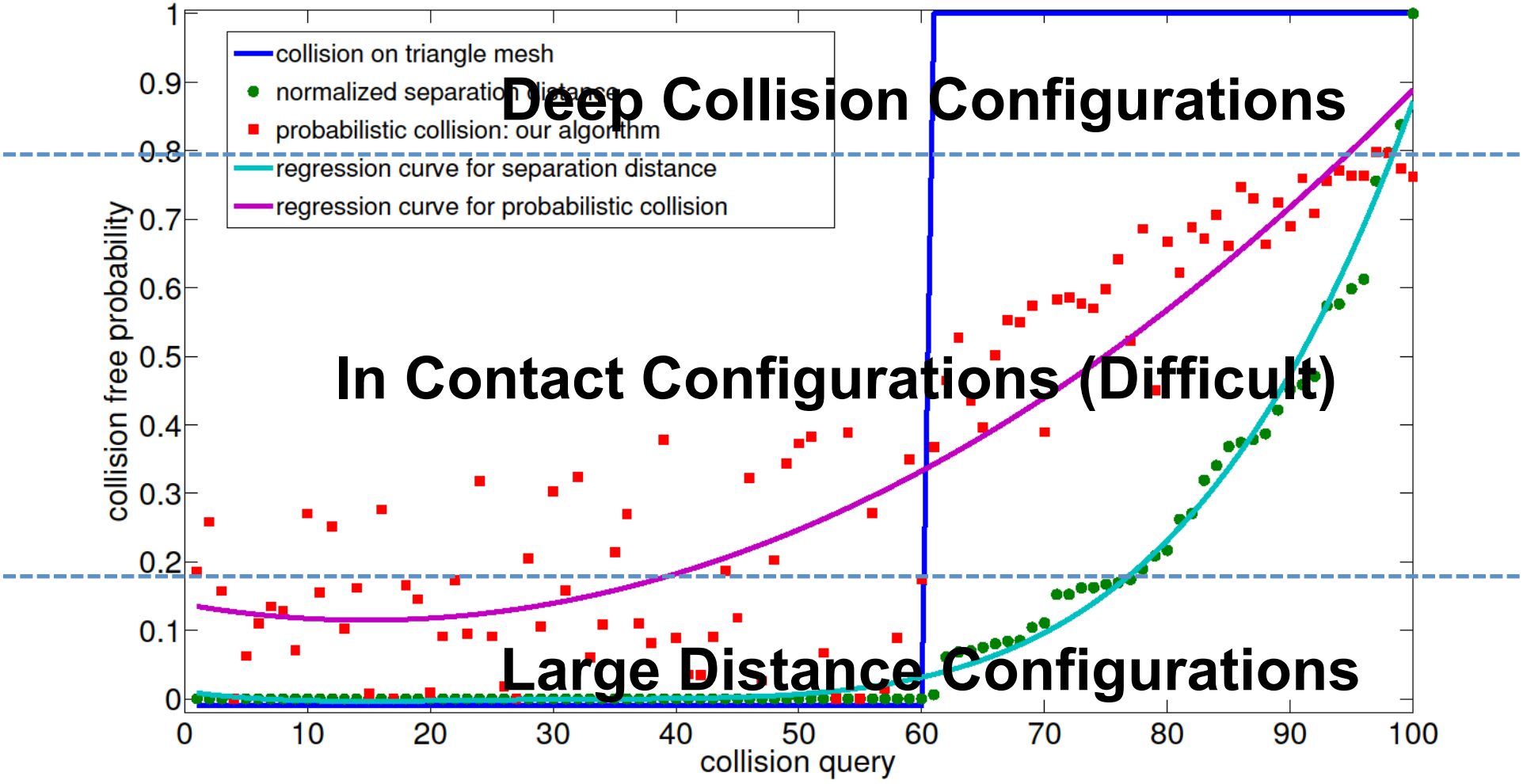
Results: Small Noise



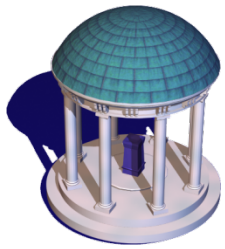
Very few configurations are in the difficult region



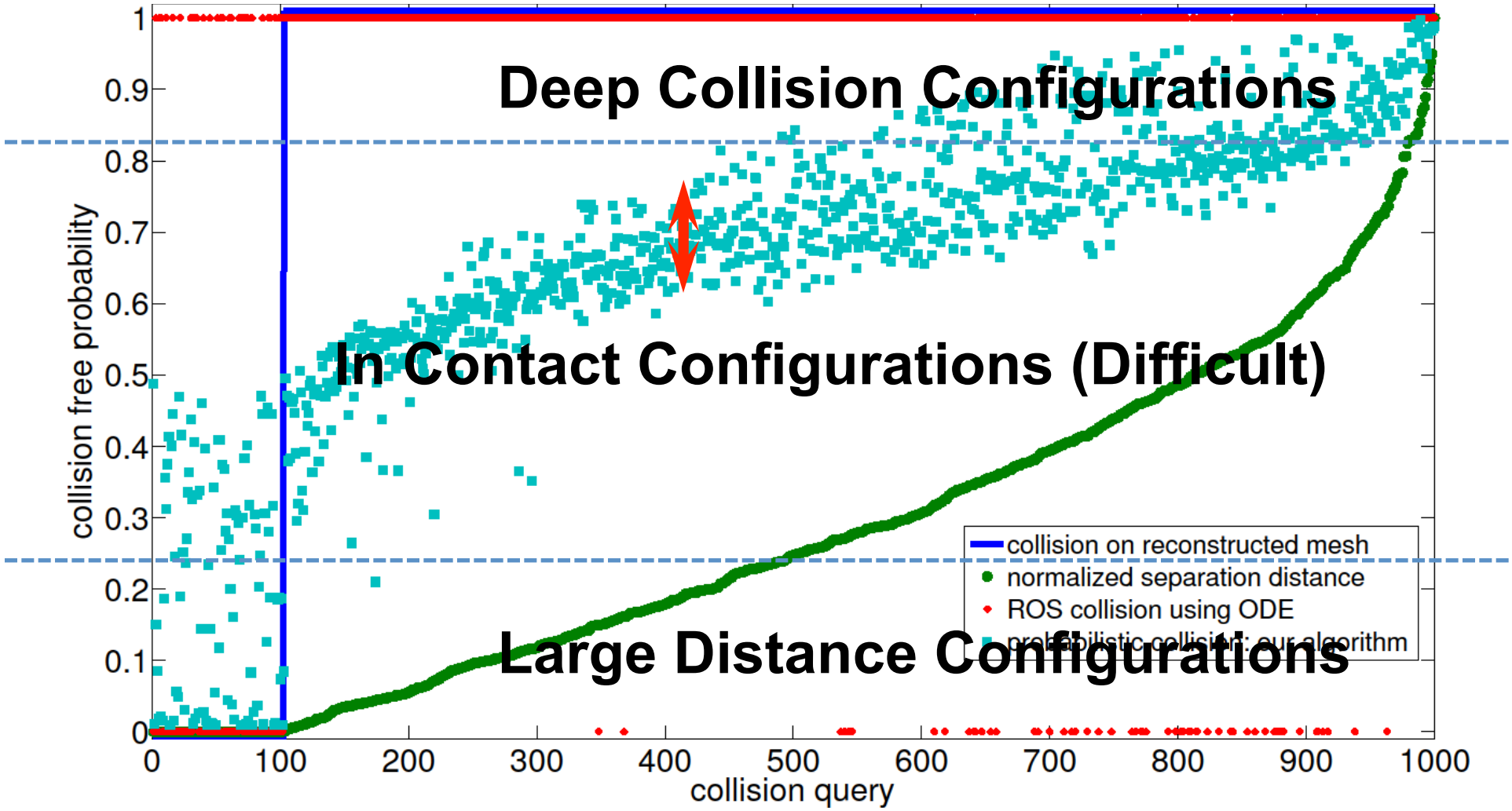
Results: Large Noise



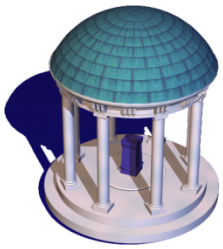
More configurations are in the difficult region!



PR2 Robot Sensor Results

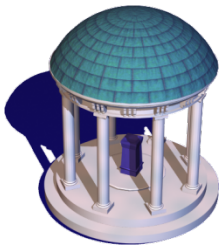


same distance to obstacle ←→collision probability with wide spread.



Benefit

- Speed is comparable to mesh collision detection algorithm (50-100ms per query) --- faster than mesh reconstruction, especially on large point clouds
- In-contact configurations corresponds to a set with non-zero measure in C-space; however, mesh-based collision checking on such objects would result in a zero-measure set



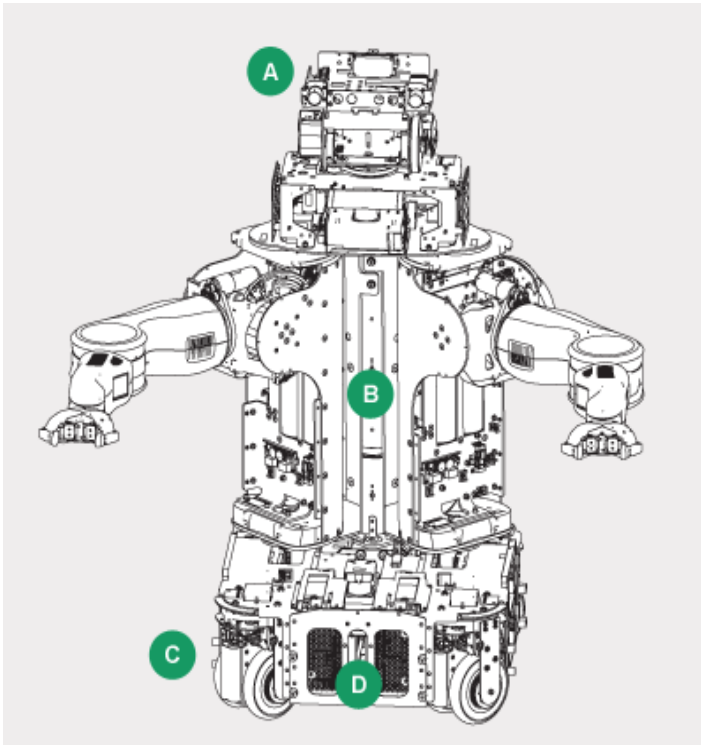
Benefit

- Speed is comparable to mesh collision detection algorithm (50-100ms per query) --- faster than mesh reconstruction, especially on large point clouds
- In-contact configurations corresponds to a set with non-zero measure in C-space; however, mesh-based collision checking on such objects would result in a zero-measure set

Motion Planning for Physical Robots

- Collision checking on noisy point cloud data
- Real-time planning using graphics hardware

Mobile Manipulators: Onboard Computation

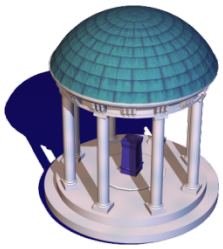


2x Onboard Servers Processors :: Two Quad-Core i7 Xeon Processors **(8 cores**)

Memory :: 24 GB

Externally Removable Hard Drive :: 1.5 TB

Internal Hard Drive :: 500 GB

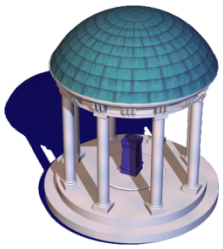


PR2 Computing Hardware

2x

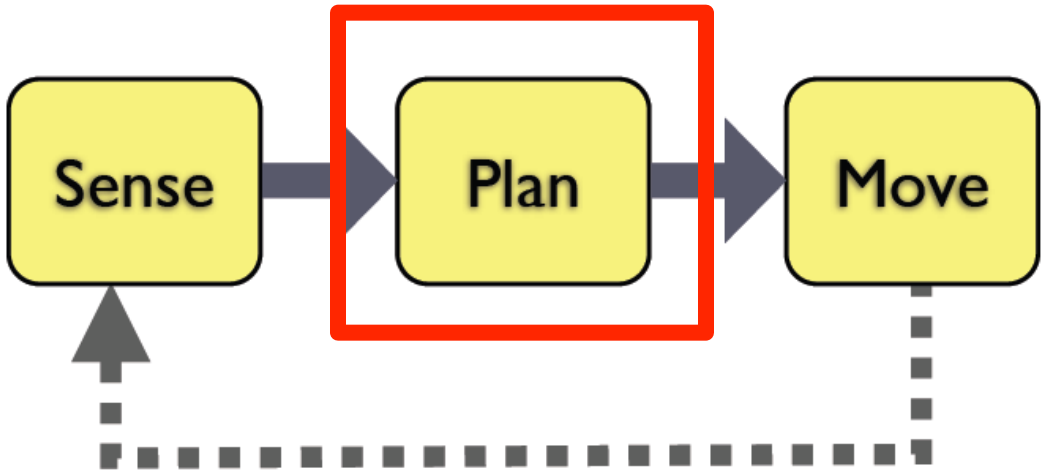
2x Quad Core Xeon 17 | 24 GB DDR3 ECC Memory | 500 GB Internal HD | 1.5 TB Removable HD

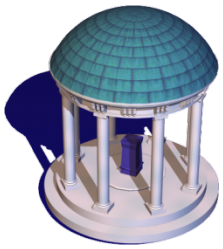
The 1.3 kWh battery system can run the servers and robot at full tilt for 2 hours (i.e. power is 650W).



Real-time Motion Planning

- Complex task execution needs *real-time feedback*
- Dynamic / uncertain / deformable environments or apply uncertain control.

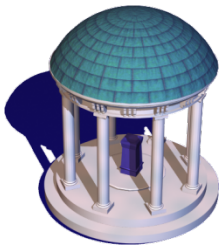




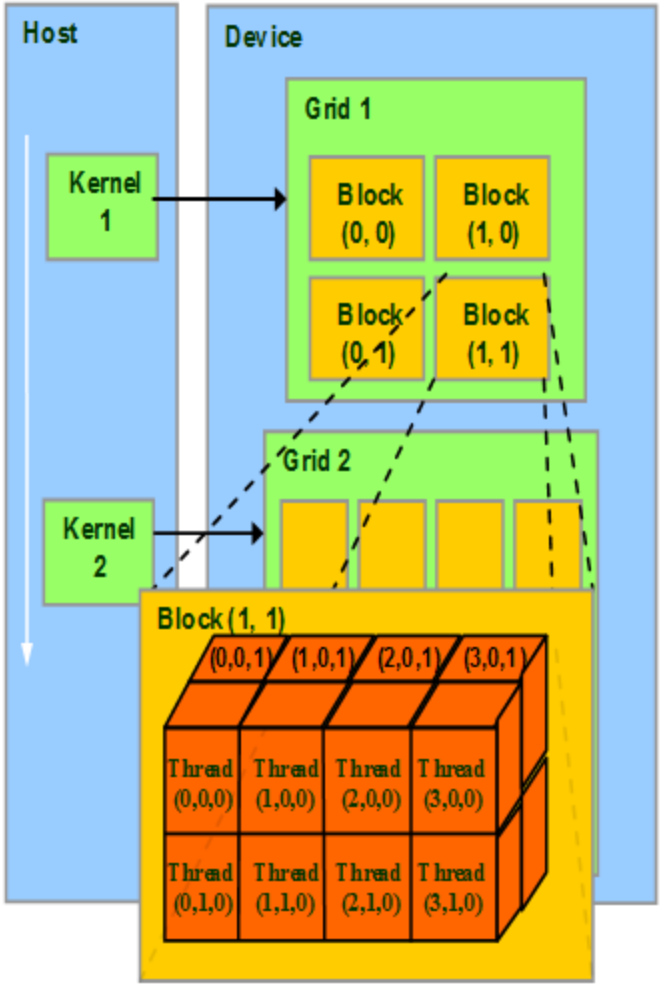
Commodity GPU

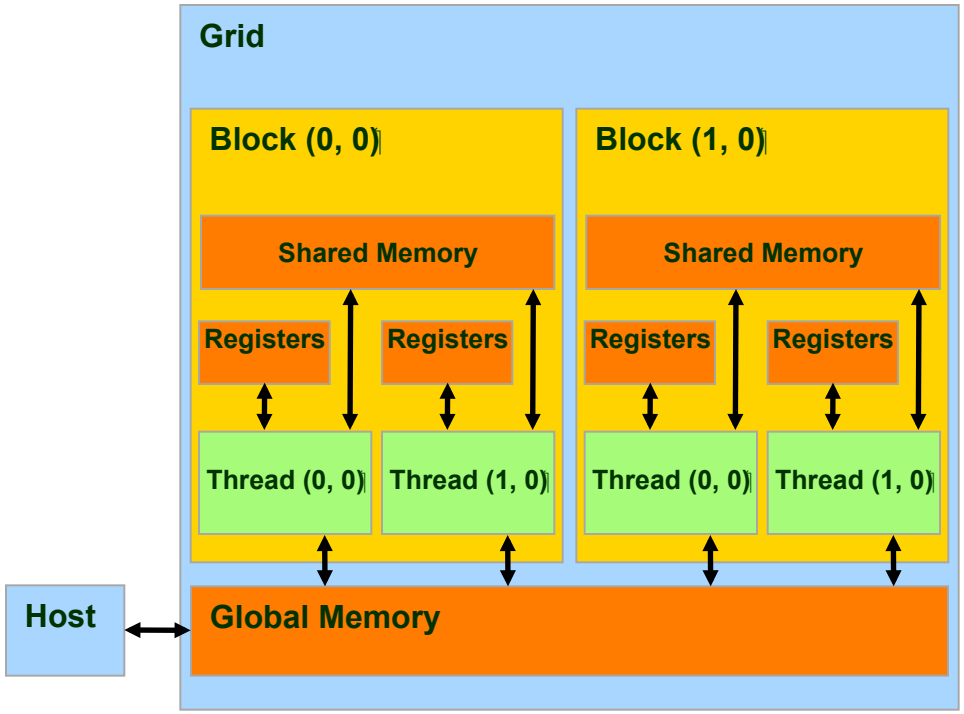
- Many-core programmable processors
  - High number of independent cores (16 - 30)
  - Wide vector units on each core (8 - 32)
  - High bandwidth, high latency main memory
  - Much higher performance/power ratio

- Synchronization between cores
  - Only via main memory
  - No memory consistency model!

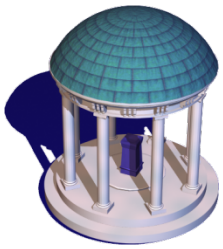


Current GPU Architectures



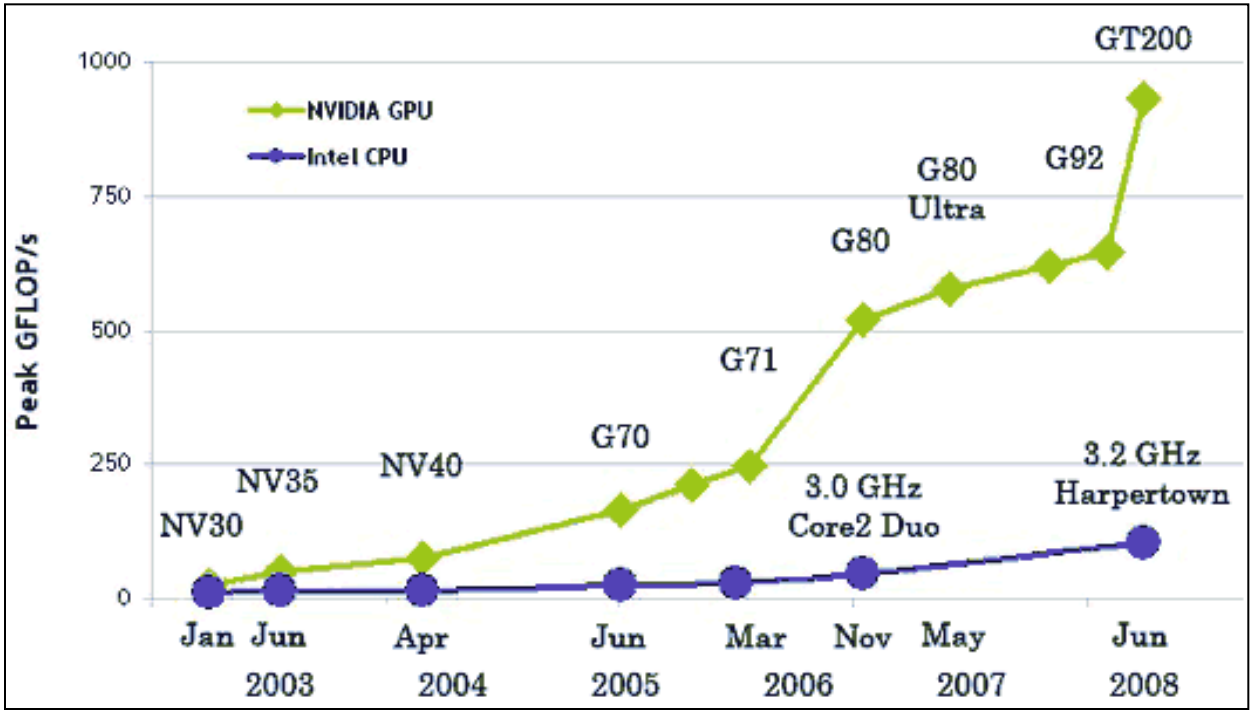


Each processor only execute one warp (32) of threads on a block. The number of parallelizable blocks is restricted by the shared memory used per block.

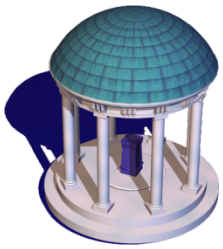


Why GPUs?

- GPUs can be faster/cheaper/smaller over CPU

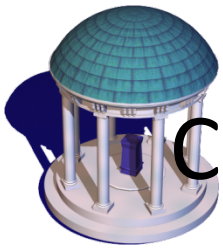


- Newest generation *Fermi* GPU can provide another 2-3 times speed-up



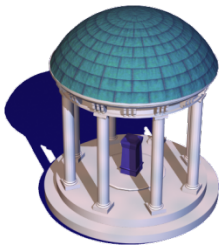
GPGPU: GPUs for non-graphics applications

- GPU has been an apparent candidate to speed-up *general purpose* large-scale computations …
  - Numerical linear algebra
  - Sorting [Owens et al. 2008]
  - Fourier Transforms [Leischener et al. 2009]
  - Acoustic Wave Equation [Mehra et al. 2010]
  - Delayed Duplicate Detection for memory management [Edelkamp et al. 2010]
  - Search [Joseph Kider et al. 2010]
  - Database query processing [Govindaraju et al. 2004; He et al. 2009]
  - Low degree-of-freedom motion planning [Hoff et al. 2000; Pisula et al. 2000]



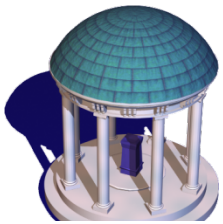
Challenge for Real-time Planner on GPUs

- Algorithmic bottlenecks
  - Algorithm complexity is high w.r.t. DOFs and topology of configuration space
- Architecture restrictions
  - GPU is not an ideal Parallel Random Access Machine (PRAM) Processor
  - Parallel planning algorithms designed for multi-core or multiple CPUs do not map well to GPU architectures

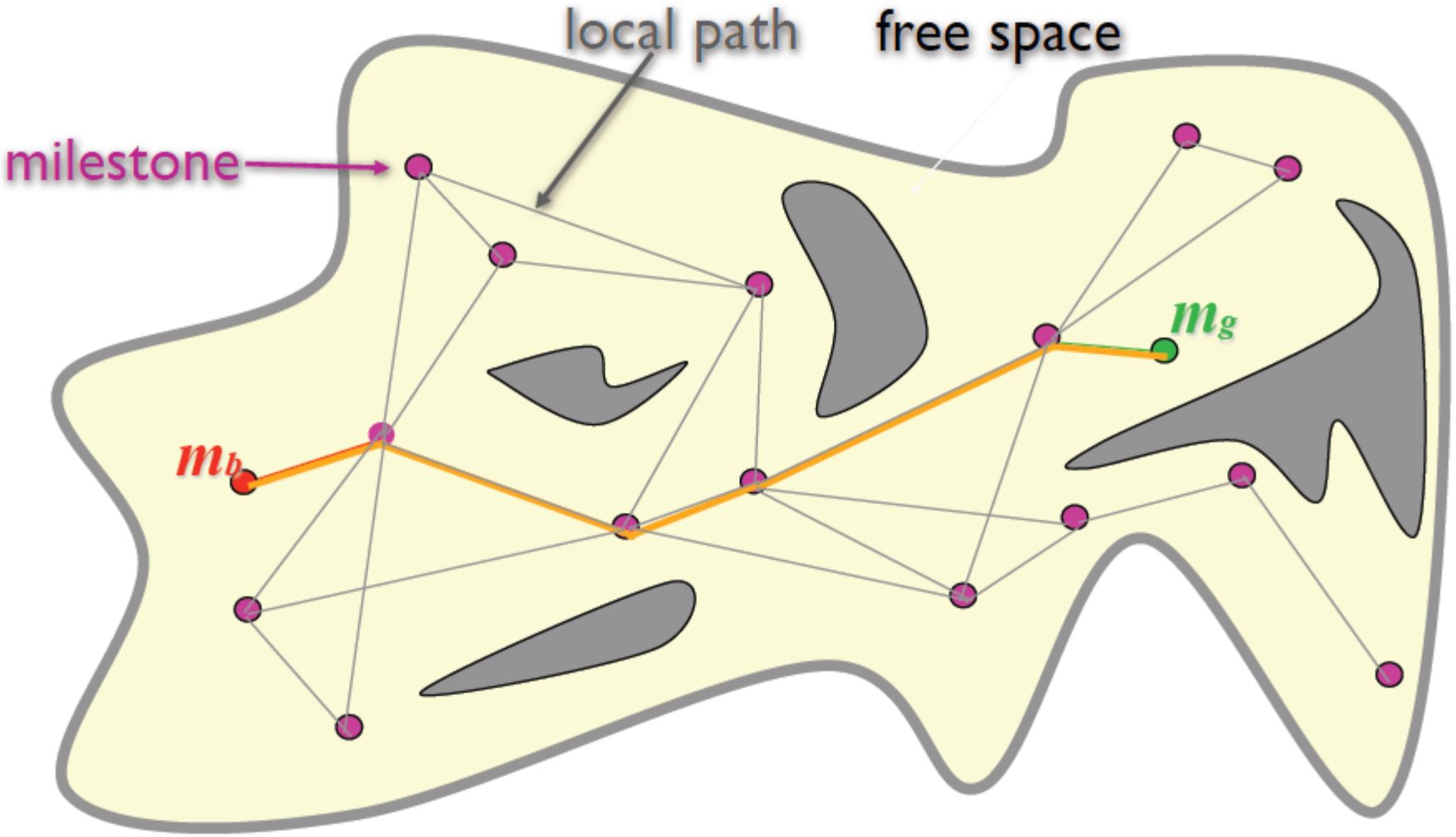


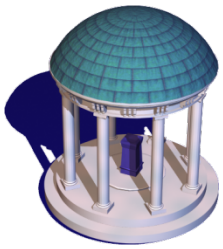
Probabilistic Roadmap Method

- Use GPUs for realtime motion planning (g-Planner)
- *g-Planner* uses probabilistic roadmap method (PRM) as the underlying motion planning algorithm
  - suitable to exploit the multiple cores and data parallelism on GPUs
  - Easier to extend for handling uncertainty

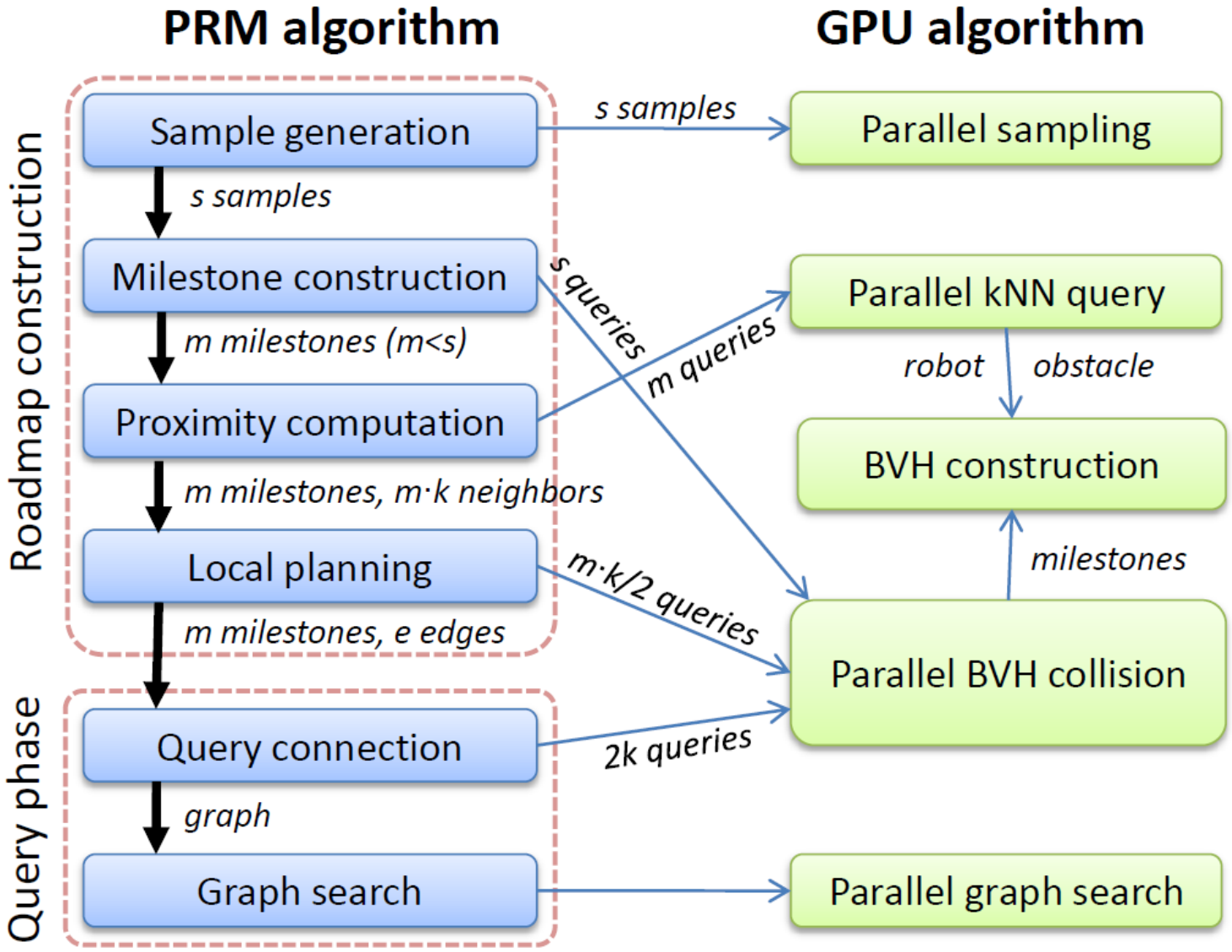


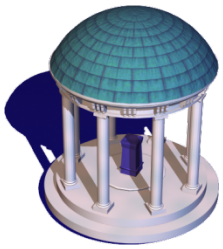
Probabilistic Roadmap Method



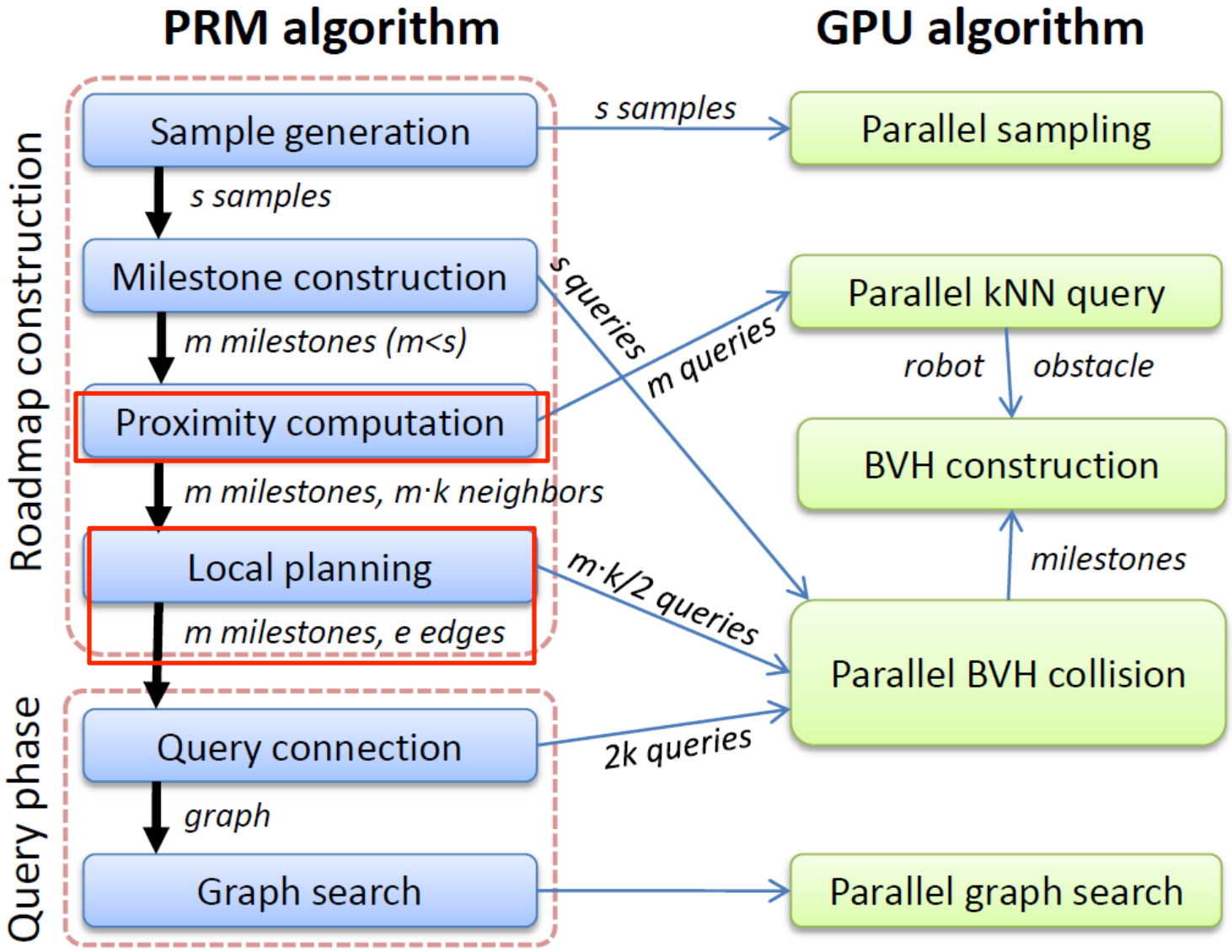


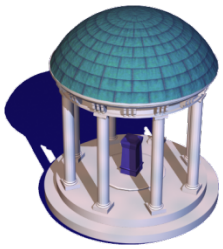
Our GPU-based Pipeline





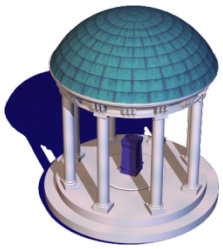
Bottlenecks in Motion Planner



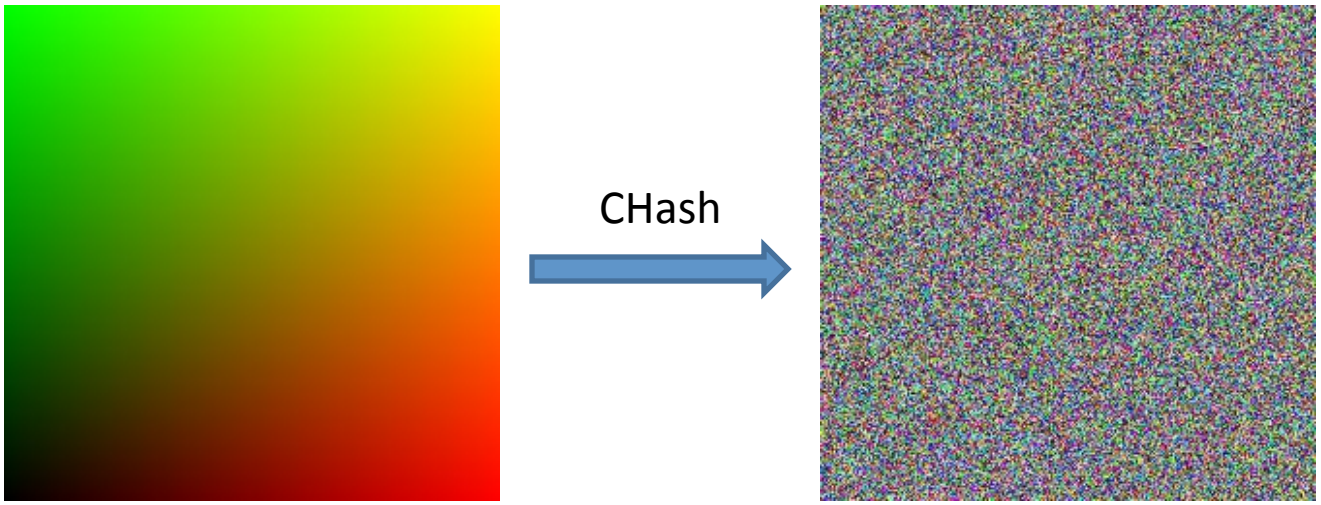


Parallel Sampling

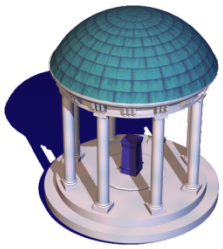
- Real parallel random generator based on cryptographic hashing [Tzeng et al. 2007]:
  - No internal state (i.e. value at i + 1 does not depend on value at i)
- Properties of cryptographic hashing
  - decorrelating – similar inputs, dissimilar outputs
  - uniform probability – all outputs likely to occur
- CHash(thread id) – white noise generator



Parallel Sampling

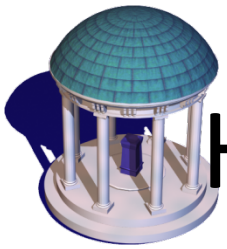


- Transformations can be performed independently on uniform samples in each thread



Bottlenecks

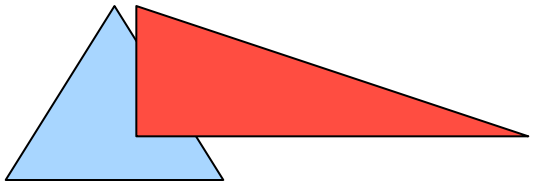
- High number of collision queries
  - Compute milestones and local planning
  - Need more than 1,000,000 even for simple roadmaps
- K-nearest neighbor query
  - Difficult when number of samples is large

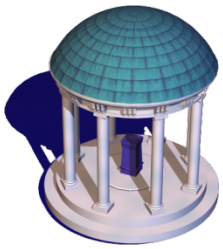


Hierarchy-based Collision Query

- Build or update hierarchies
- Traverse hierarchies recursively
  - Start with root nodes
  - Do nodes overlap?
    - **Yes:** Inner nodes: recursive formulation
    - Perform primitive overlap tests

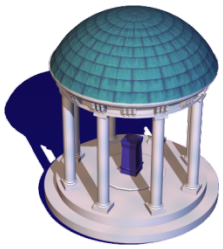
Triangle-triangle overlap test





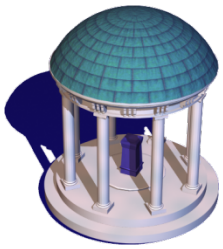
BVH Construction

- Construct BVH on GPU
  - Uses thread and data parallelism
  - Fast linear BVH construction [Lauterbach et al. 2009]
- Basic idea: turn the BVH construction problem into the *sorting* problem along a space-filling curve (i.e. Morton Curve)
  - Points close on curve are close in 3D space
  - Radix-sort is fast on GPU
- Interactive construction on current GPUs (<30ms)



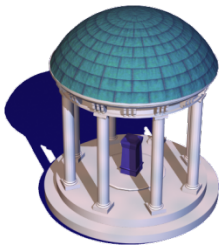
Parallel Collision Query

- In motion planning, we can perform high number of collision queries in parallel
- Naive parallelism is used in previous CPU-based parallel planner [Akinc et al. 2005, Amato et al. 1999]
  - Per-thread per collision or per-thread per continuous collision

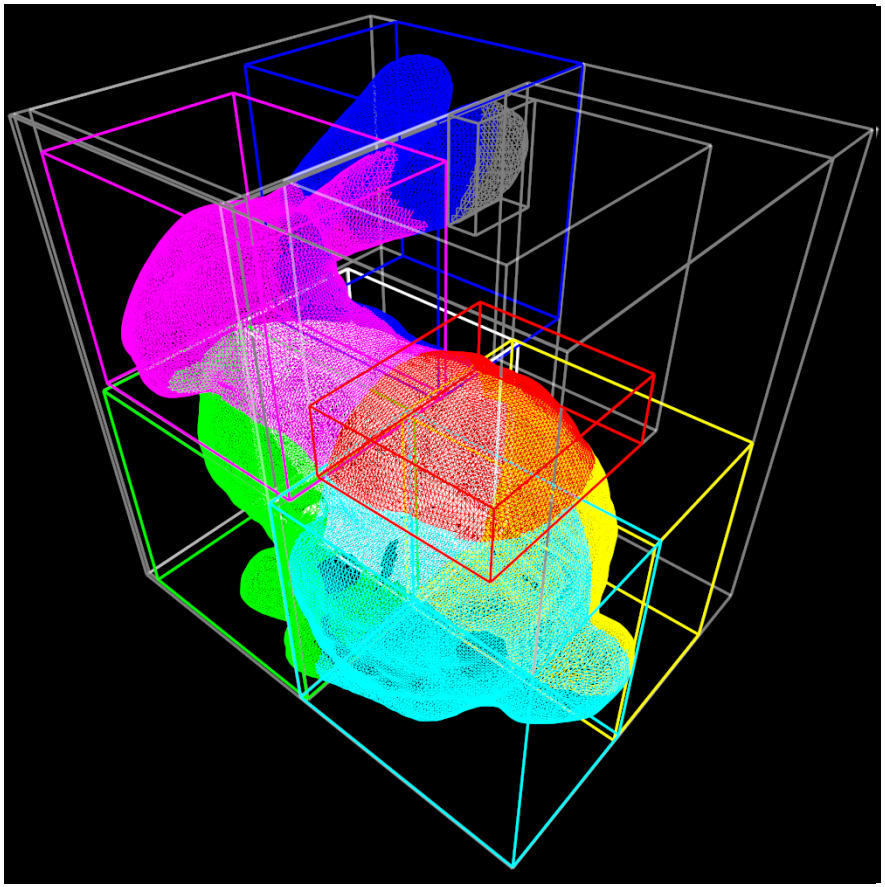


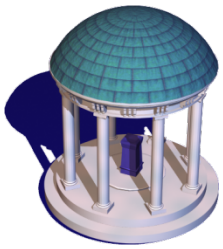
GPU-based Parallel Algorithms

- Collision queries
  - Use Bounding Volume Hierarchy (BVH) for acceleration
  - Handle multiple samples in parallel
  - Lazy planning: delay collision queries in local planning until necessary
- K-nearest neighbor search
  - Locality-sensitive hashing based approach
  - Turn global search into local search
  - Linear complexity



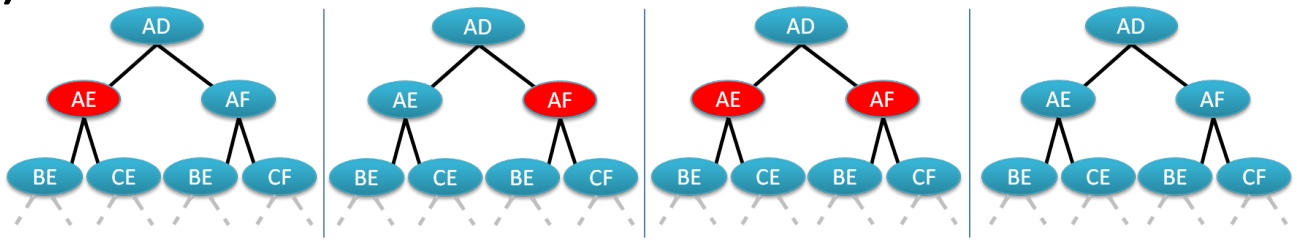
Hierarchies

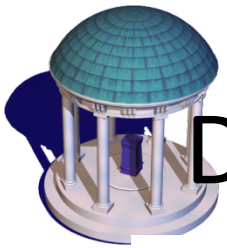




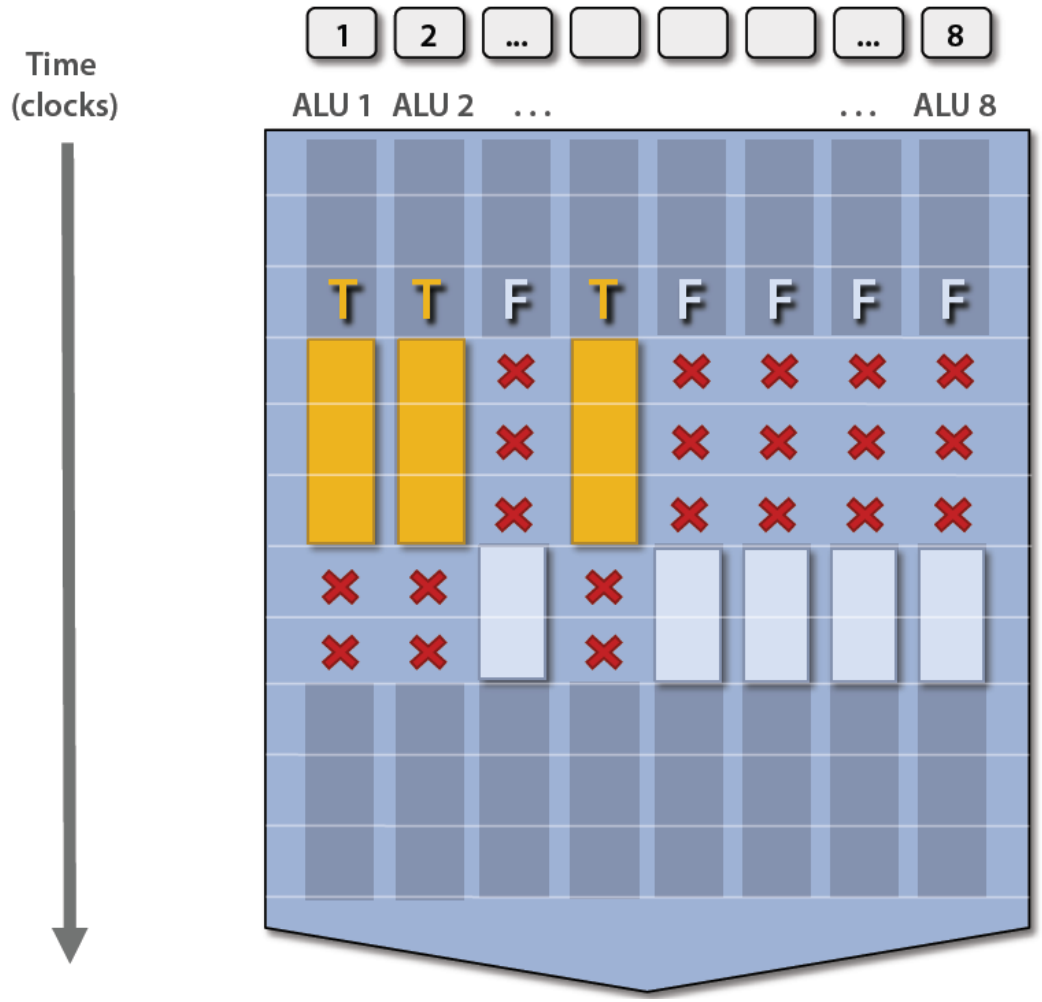
GPU Memory Model

- Shared memory is fast, BUT limited (16K-48K)
- The more shared memory used for one block, the less parallelism
  - Parallel block num ≤ $\frac{\text{overall shared memory size}}{\text{shared memory used per block}}$
  - Basic parallel BVH algorithm needs one stack (>32) for each thread, so 32 * n for a n-thread block (BAD!)





Data-dependent Conditional Branch



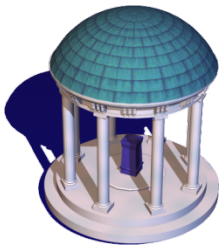
(Per warp)

```
<unconditional
shader code>

if (x > 0) {
    y = pow(x, exp);
    y *= Ks;
    refl = y + Ka;
} else {
    x = 0;
    refl = Ka;
}

<resume unconditional
shader code>
```

Happens frequently in BVH traverse (BAD!)

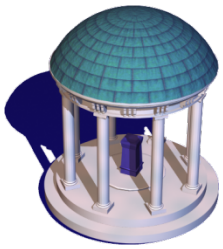


Our Solutions

- Parallel Collision-Packet Traversal
  - 50%-100% speed up over basic GPU method
  - Simple to implement and can be used with basic parallel collision algorithms
- Parallel Collision Query with Workload Balancing
  - 5-10x speed up over basic GPU method
  - More complicated to implement

Parallel Collision-Packet Traversal

- Cluster collision queries into several groups
  - Queries in one group will have similar traverse path
  - Groups are further divided into small warps
- For queries in the same warp traverse the BVTT in the same special order
  - One stack per block
  - Coalesced memory access and cacheable
  - No branch divergence



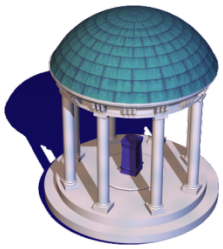
Query Clustering

- Find $K = \lceil \frac{N}{chunk\text{-}size} \rceil$ clusters to minimize

$$\sum_{i=1}^{N} \sum_{k=1}^{K} \mathbf{1}_{\mathbf{x}_i \in C_k} \|\mathbf{x}_i - \mathbf{c}_k\|$$

where $\{\mathbf{c}_k\}_{k=1}^{K}$ are cluster centers and clusters are of the size $|C_k| = chunk\text{-}size, 1 \leq k \leq K$

- Constrained clustering, difficult to solve
- We only approximate it with k-means and then divide into chunk-size clusters.



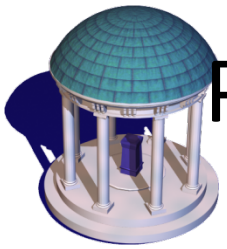
Packet's Traverse Order

- We need an optimal traverse order for the packet to avoid additional BV collisions.
- Notice that only the entire packet do not overlap with one BV can the packet stop.
- Using greedy heuristics
  - The probability for one traverse order P is

$$p_P = \prod_{(x,y)\in P} p_{x,y}$$

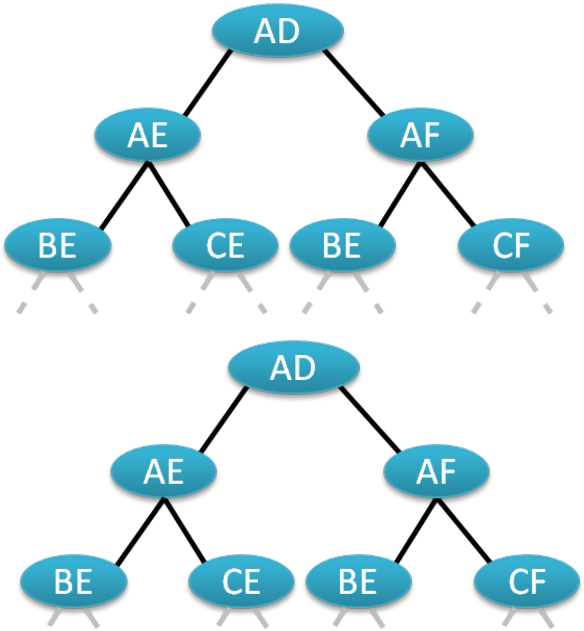
where $p_{x,y} = \frac{\#overlap\ threads}{packet\text{-}size}$

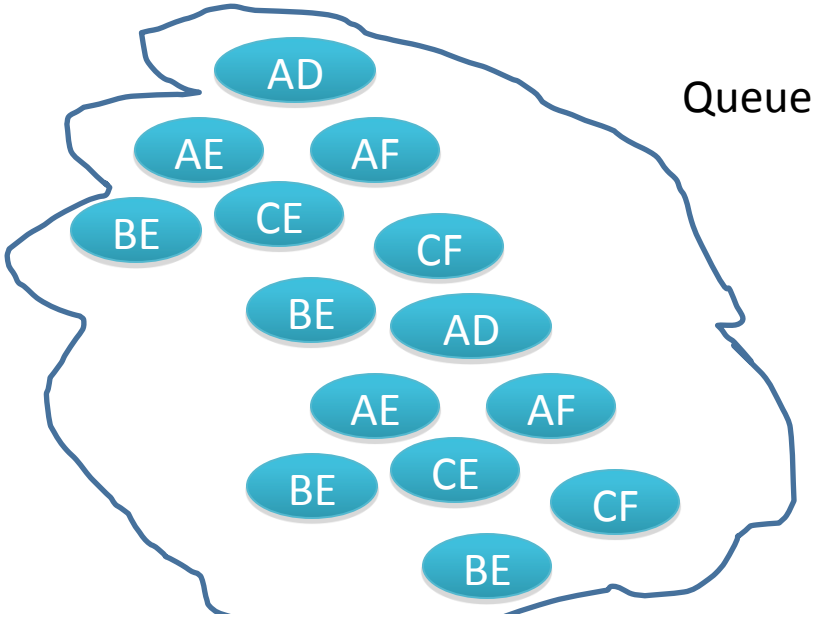
  - Traverse the children node with large $p_{x,y}$ first

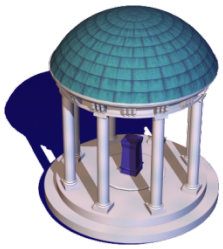


Parallel Collision Query with Workload Balancing

- Each thread executes more fine-grained tasks: overlap test between two BVs or leaf triangles
- The tasks are stored in one large queue, and keep a local task queue for each block.

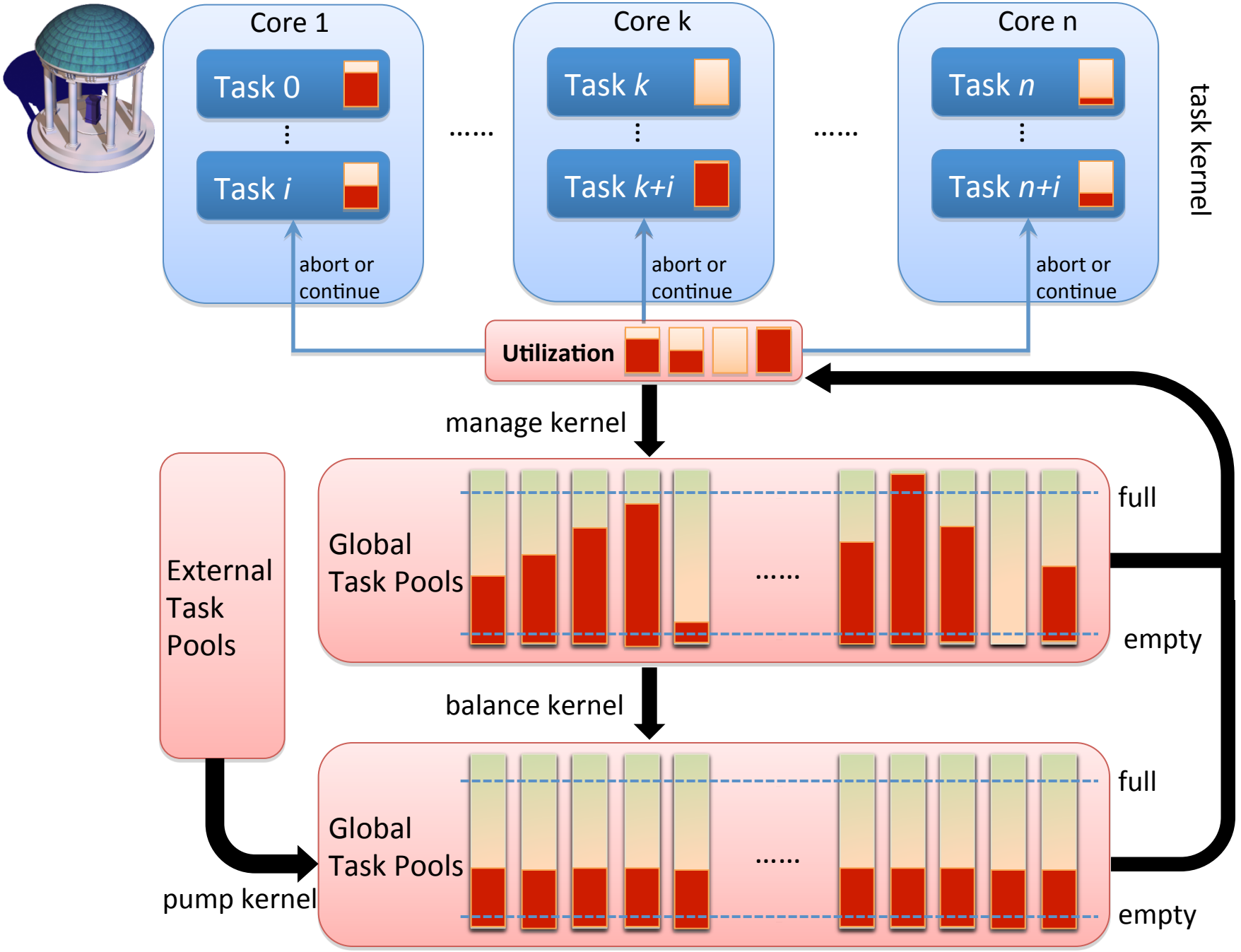


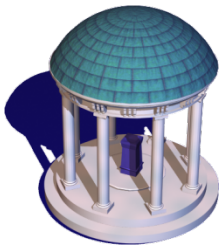




Workload Balancing

- Different queries will stop after different number of BV checks
- Different local queues will have different number of tasks
- Queue is nearly full or empty→ processor idle → balancing


Core 1
Task 0
Task i
Core k
Task k
Task k+i
Core n
Task n
Task n+i
task kernel
abort or continue
abort or continue
abort or continue
Utilization
manage kernel
External Task Pools
Global Task Pools
full
empty
balance kernel
Global Task Pools
full
empty
pump kernel



Performance Analysis

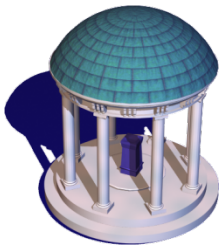
- We can prove that our parallel algorithms on GPU are **work efficient**, i.e. not slower than the serial implementation.
- $T_{serial} > T_{basic} > T_{packet} > T_{workload} \approx T_{serial}/\#processor$

T_{serial} $T_S(n) = \sum_{i=1}^{n} W(i) \geq \sum_{k=1}^{a} W^*(kp)$

T_{basic} $\sum_{i=1}^{n} W(i) \geq T_N(n) \geq \frac{\sum_{i=1}^{n} W(i)}{p}$

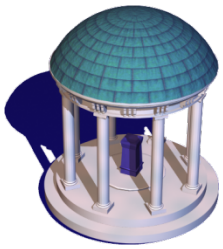
T_{packet} $T_P(n) \approx \sum_{k=1}^{\hat{a}} \hat{W}^*(kp)$, with $\hat{W}^* \leq W^*$

$T_{workload}$ $T_B(n) = \frac{\sum_{i=1}^{n} W(i)}{p} + B(n)$



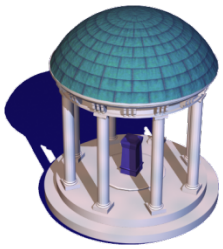
K-nearest neighbor computation

- Many previous approaches are based on spatial structures, e.g. Kd-tree, GNAT, BVH
  - Worst space/time complexity is square
  - Complexity grows exponentially according to the dimensionality
- Our solution
  - Based on locality-sensitive hashing (LSH) and cuckoo hashing
  - Turn global KNN search into local search
  - Linear complexity



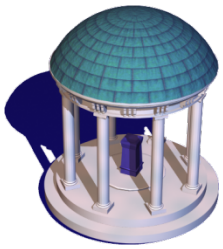
Hashing-based KNN search

- Approximate method
  - Approximated KNN is sufficient roadmap construction
- Basic idea
  - LSH: group samples potentially to be closed
  - Cuckoo hashing: efficient storage and query
- At least 10x speed-up (better for large dataset and high-dimensional data)

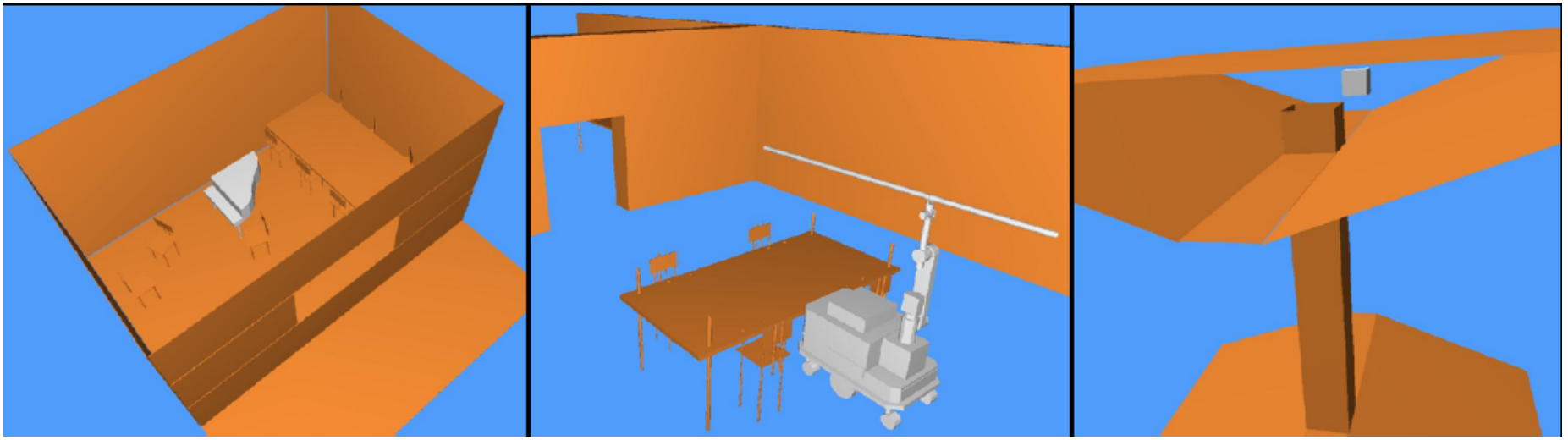


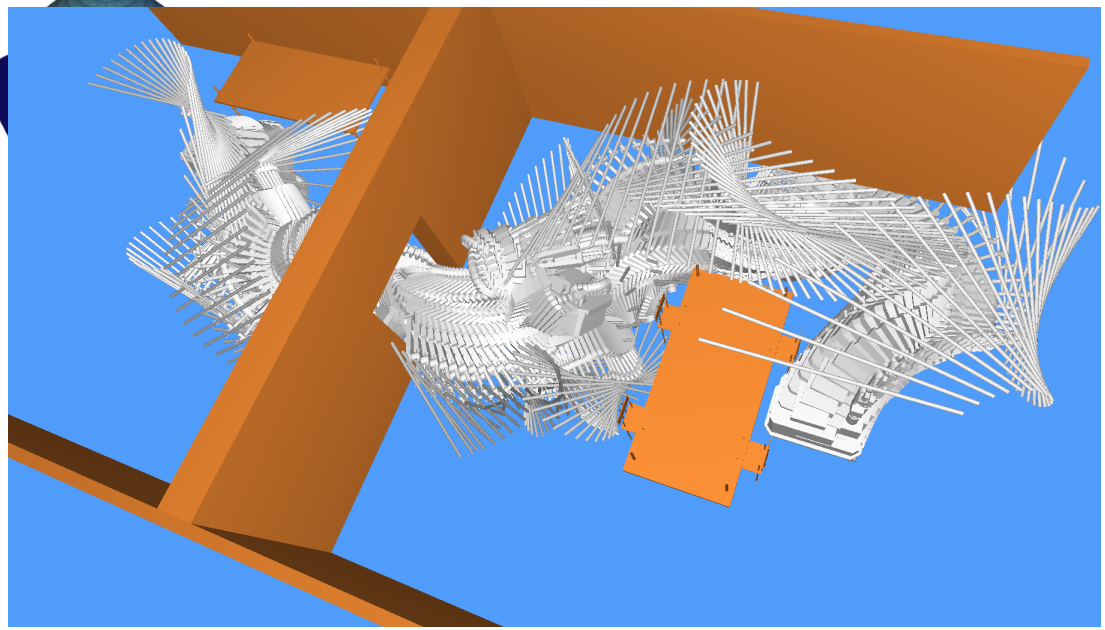
Graph Search

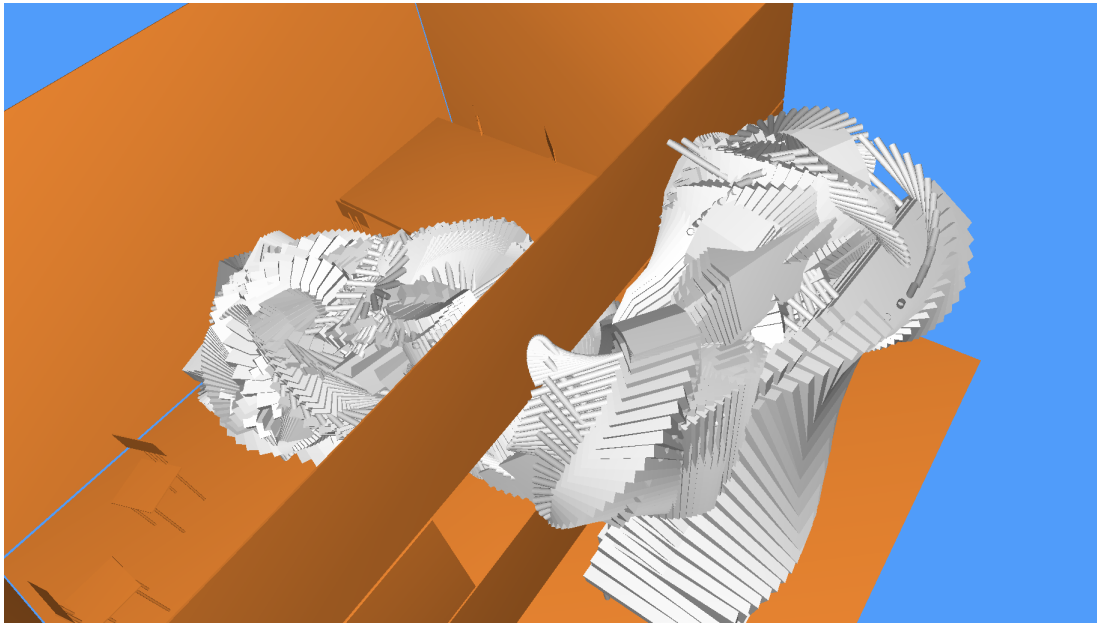
- Currently perform parallel BFS/DFS
- A* or more advanced method (ARA* etc) can be used to improve the search [Kider et al. 2010]

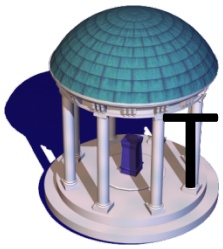


Benchmarks





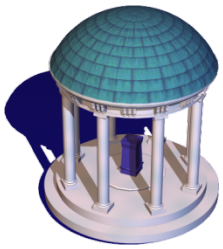




Timing Results: Collision Checking

- Compared with basic GPU method [Lauterbach et al. 2010]

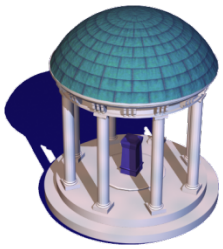
50000 collision queries	Basic GPU	Collision-packet	Workload Balancing	
			Traverse	balance
Piano	224	130	68	3.7
Large-piano	710	529	155	15.1
Helicopter	272	226	56	2.3
Humanoid	2316	1823	337	126



Timing Results: Local Planning

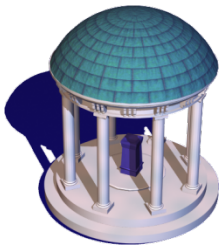
- Compared with basic GPU method (per-thread per query method) (ms)

Local Planning Computations	Basic GPU	Collision-packet	Workload Balancing	
			Traverse	balance
Piano	2076	1344	1054	34
Large-piano	7587	6091	1139	66
Helicopter	7413	4645	913	41
Humanoid	8650	8837	6082	1964



Overall Performance

- Our parallel GPU-based algorithms can perform about 500K collision queries per second on $400 NVIDIA Fermi Card (100X faster than prior methods)



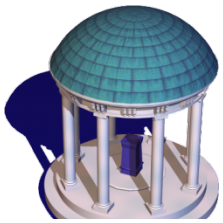
PRM Motion Planning on GPUs

- 100x acceleration can be observed

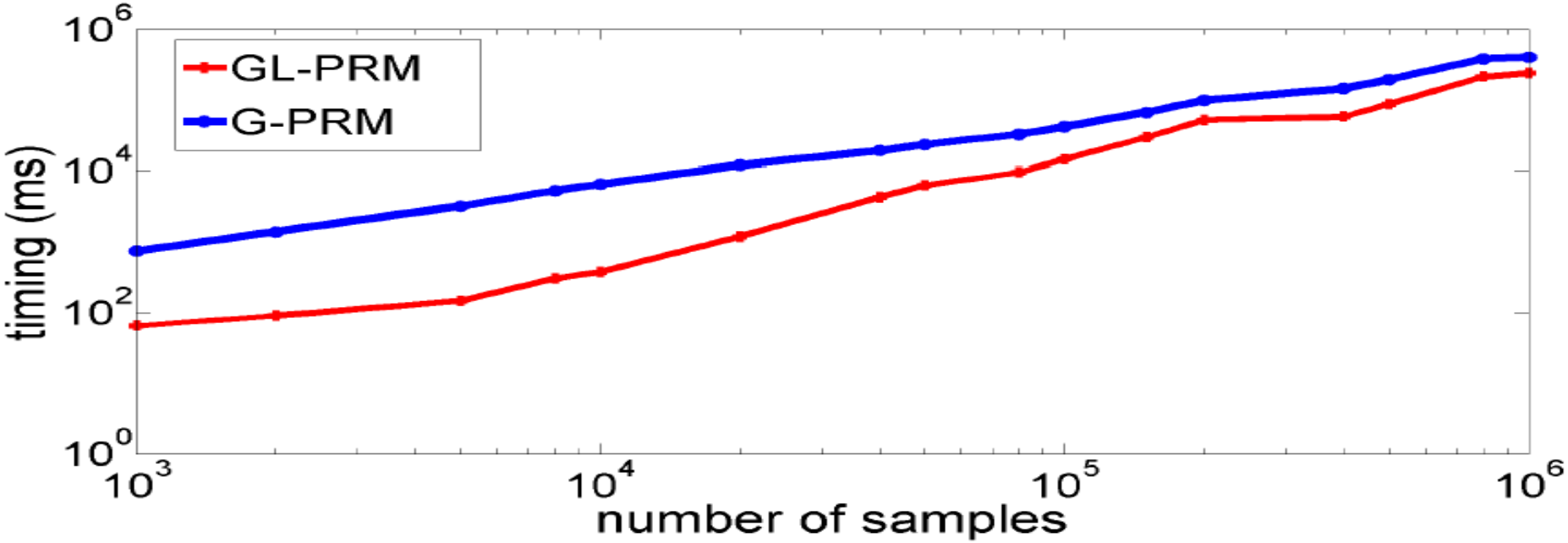
	C-PRM	C-RRT	G-PRM	GL-PRM
piano	6.53s	19.44s	1.71s	111.23ms
helicopter	8.20s	20.94s	2.22s	129.33ms
maze3d1	138s	21.18s	14.78s	71.24ms
maze3d2	69.76s	17.4s	14.47s	408.6ms
maze3d3	8.45s	4.3s	1.40s	96.37ms
alpha1.5	65.73s	2.8s	12.86s	1.446s

OOPSMP on Intel 3.2GHz i7 (single core) CPU ($600)

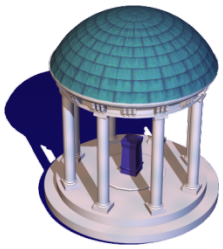
gPlanner on NVIDIA GTX 285 GPU ($400)



Preliminary Results

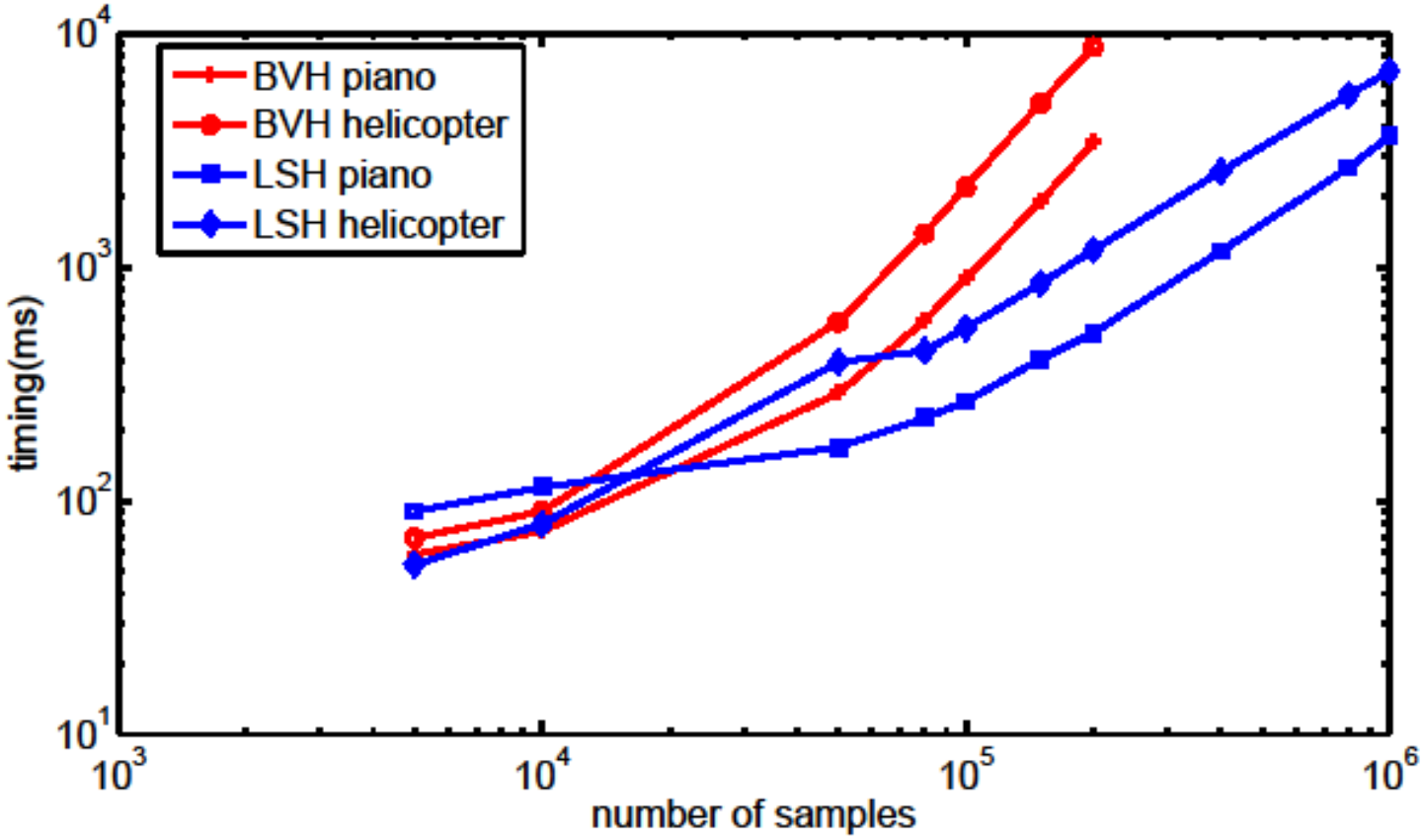


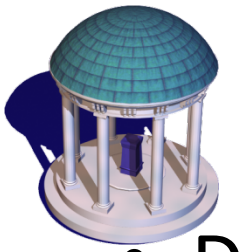
- Scale well on multi-core GPUs (log-log plot)



Preliminary Results

- Scalability of LSH based KNNS (log-log plots)



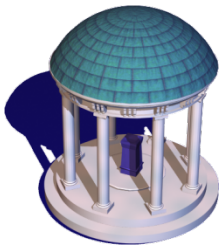


Applications to PR2 Model

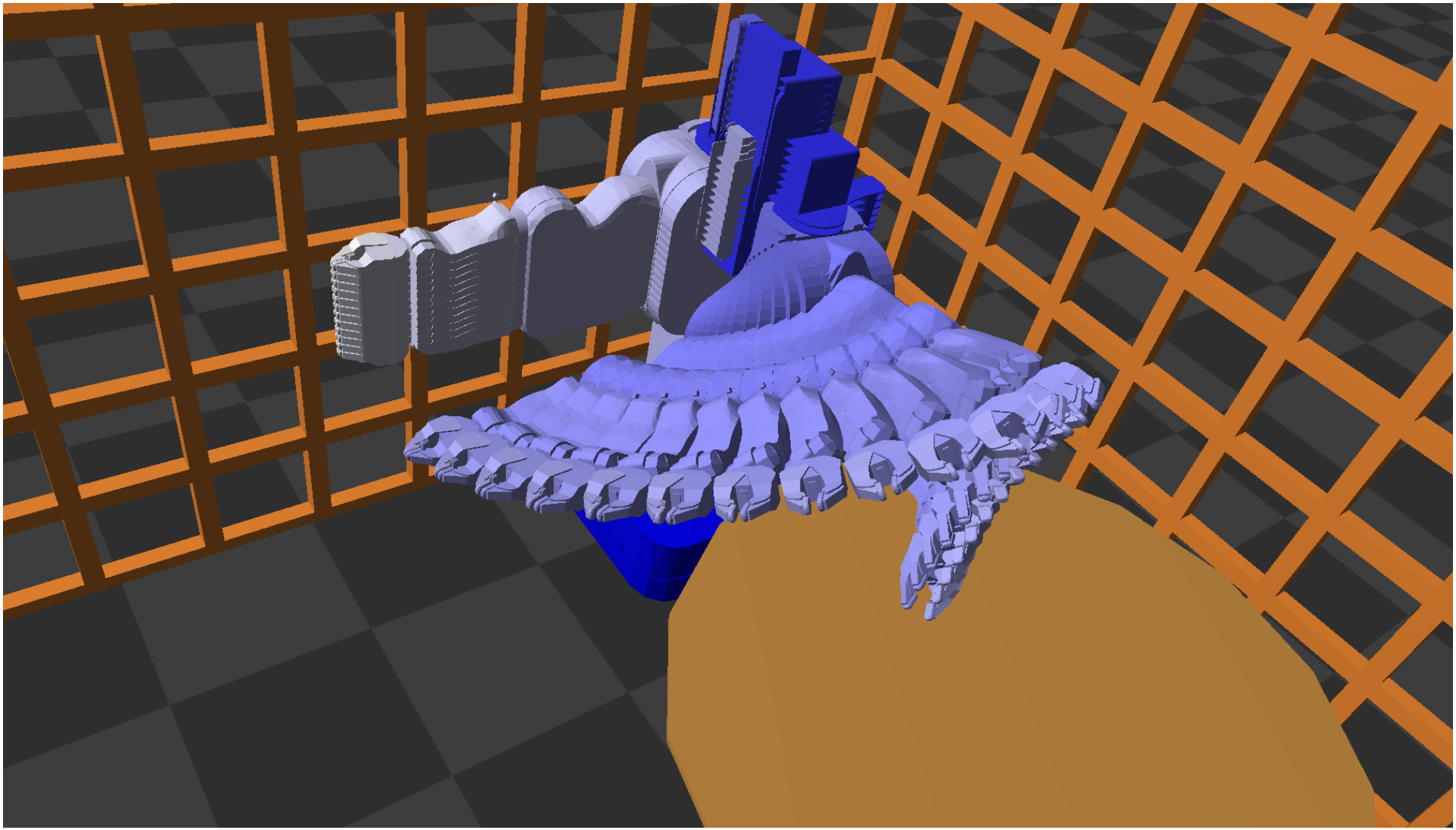
- DOF 12
- Compared with CPU (ms)

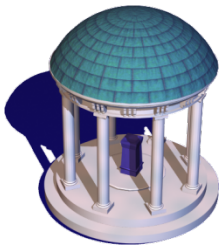
	CPU	GPU
Milestone Comp.	15,952	392
Local Planning (include self-collision)	643,194	6,803

- 500 samples
  - Perform motion planning in simple scenarios in ~300ms

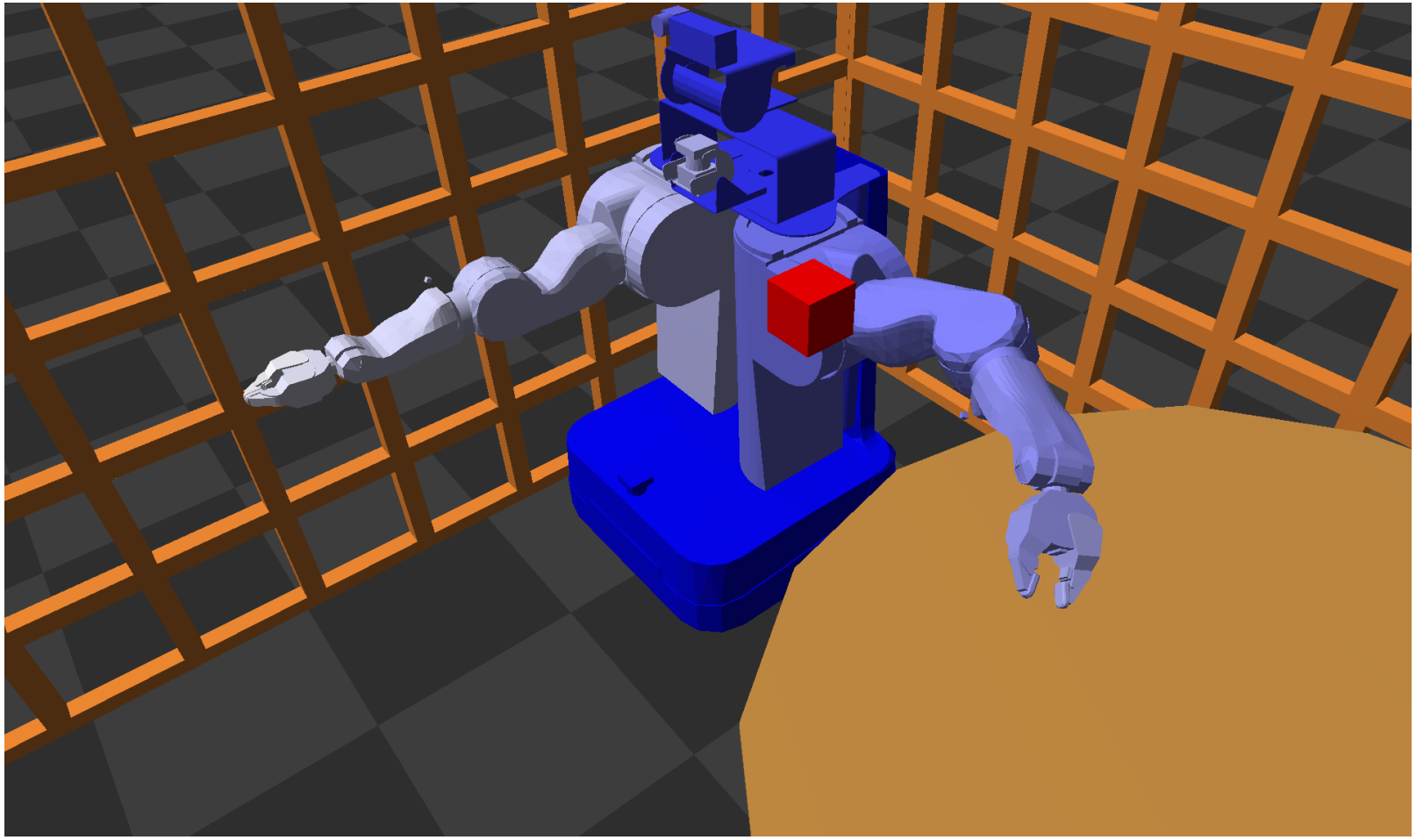


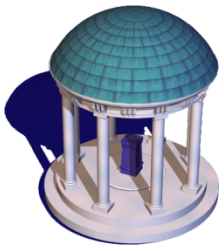
Results





Results: PR2 Model

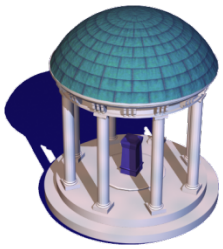




GPU-based Motion Planning

- Multi-core GPUs' computational power make real-time planning possible
- Suitable parallel algorithms need to design to achieve peak performance on the specific architecture of GPUs

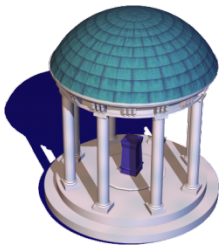
- GPUs can be exploited for a variety of search problems



Conclusions & Future Work

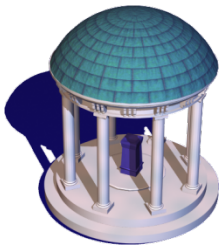
- Collision checking on noisy point cloud data
- Real-time planning using graphics hardware

- Integrate with Physical Robots (PR2)



Acknowledgements

- Funding agencies
  - ARO
  - NSF
  - DRAPA/RDECOM
  - Intel



Thanks