High-Dimensional Planning on the GPU
Mark Henderson, Joseph T. Kider Jr., Maxim Likhachev, and Alla Safonova
SIG Center for Computer Graphics, University of Pennsylvania
mahee@seas.upenn.edu, kiderji@seas.upenn.edu, maxmil@seas.upenn.edu, alla@seas.upenn.edu

Abstract
Optimal heuristic searches such as A* search are commonly used for low-dimensional planning such as 2D path finding. These algorithms however, typically do not scale well to high-dimensional planning problems such as motion planning for robotic arms, computing motion trajectories for non-holonomic robotic vehicles and motion synthesis for humanoid characters. A recently developed randomized version of A* search, called R* search, scales to higher-dimensional planning problems by trading off deterministic optimality guarantees of A* for probabilistic sub-optimality guarantees. In this paper, we show that in addition to its scalability, R* lends itself well to parallel implementation. In particular, we demonstrate how R* can be implemented on GPU. On the theoretical side, the GPU version of R*, called R*GPU, preserves all the theoretical properties of R* including its probabilistic bounds on sub-optimality. On the experimental side, we show that R*GPU consistently produces lower cost solutions, scales better in terms of memory, and runs faster than R*. These results hold for both motion planning for 6DOF robotic arm as well simple 2D path finding shown by our detailed experimental analysis section.

Parallelization of R* Search (R*GPU)
• It turns out that the decomposition of a single-shot search into a series of easy-to-solve short-range searches lends itself naturally to a parallel implementation on GPU.
• While the main loop (figuring out what short-range search to run next) can run on GPU, each of the short-range searches can run on a thread in CUDA.
• Each short-range search is independent of others which makes it suitable for running them in parallel.
• Each search does not require vast amounts of memory since by definition it is easy to solve.
• Allows for multiple searches to share states in the DRAM on the GPU so there are no unnecessary expansions.
• Removed need for expensive hashing functions by checking if the location has been searched to in the array then selectively overwriting cells when needed.
• This reduces the divergent branches inherent in hashing functions.

R* Search Algorithm
• R* search operates by decomposing the usual single-shot A* search into a series of properly-scheduled short-range and easy-to-solve searches, each guided by the heuristic function towards a randomly chosen goal.
• R* constructs a small graph Γ of sparsely placed states, connected to each other via edges.
• R* constructs Γ in such a way as to provide explicit minimization of the solution cost and probabilistic guarantees on the suboptimality of the solution.
• R* grows Γ the same way A* grows a search tree
• At every iteration, R* selects the next state s to expand from Γ
• While normal A* expands s by generating all of its immediate successors, R* expands s by generating K random states residing at some domain-dependent distance Δ from s.
• If a goal state is within Δ from state s then it is also generated as the successor of s. R* grows Γ by adding these successors of s and edges from s to them.
• R* postpones finding these hard-to-solve paths until necessary and concentrates on finding the paths that are easy-to-solve instead.
• R* uses the (short-range) weighted A* searches with heuristics inflated by ε > 1 to compute these easy-to-solve paths.

R* GPU Pseudocode
1. choose root state s 2. generate a random select from state s 3. while goal is not reached 4. remove entrance s, deadly from heap 5. select successors to Γ 6. generate random states for all successors 7. call weighted A* search from Γ 8. if goal has been reached 9. return 10. else 11. add states from randomly to 12. end while 13. return high level path from root to goal 14. end while

2D Planning
- Example of 2D planning scenario for 24-connected grid-world (200x200 cells)
- 3 different values for epsilon (2, 1.5, 1)

Detailed Experimental Results
• 90 randomly generated 2D grid worlds of varying obstacle density for fast (simple) and artificially time consuming (hard) edge cost expansions
• 53 randomly generated high dimensional 6 degree of freedom robotic arm tested with 3 settings of ε (Resulting State Space is over 3 billion states)
• R*GPU outperforms CPU version of R* as obstacle density grows and cost computation becomes time consuming

Related Work
• GPU Accelerated Path Finding [Bleiweiss 08]
  • Optimally Navigate Agents
  • Planning for 2D environments
  • Exploits explicit Parallelism of multi-agent navigation planning
• A* Search kernels for many agents

Experimental Results on a Simulated Robotic Arm

GPU Cost = 78

R* Search [Likhachev + Stentz 08] – Randomized Version of A*
• Depends less on the quality of guidance of the heuristic function
• Solves more high dimensional robot arm maps
• Struggles to solve maps with high object density

Future Work
• Work on an actual robotic arm to solve real time motion planning problems
• Expand for even higher dimensional joint configurations and higher dimensional spaces to make the algorithm even more robust
• Apply to high dimensional planning for human animation locomotion planning
• Robust performance comparison to multi-core and Larrabee implementations of our parallel algorithm