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, kiderj@seas.upenn.edu, maximl@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 a 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 robot arm as well simple 2D path finding shown by our detailed experimental analysis section.

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* Pseudocode

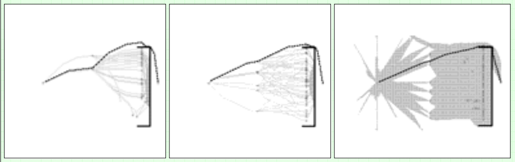
select unexpanded state $s \in \Gamma$ (priority is given to states not labeled AVOID)
if path that corresponds to the edge $bp(s) \rightarrow s$ has not been computed yet
  try to compute this path
  if failed then label state s as AVOID
  else
    update $g(s)$ based on the cost of the found path and $g(bp(s))$
    if $g(s) > w\, h(s_{\text{start}}, s)$ label s as AVOID
else //expand state s (grow Γ)
  let $SUCCS(s)$ be K randomly chosen states at distance Δ from s
  if goal state within Δ, then add it to $SUCCS(s)$
  for each state $s' \in SUCCS(s)$, add s' and edge $s \rightarrow s'$ to Γ, set $bp(s') = s$

Figure : Singe iteration of R*

2D Planning

- Example of 2D planning scenario for 24-connected grid-world (200x200 cells)

- 3 different values for epsilon (2, 1.5, 1)



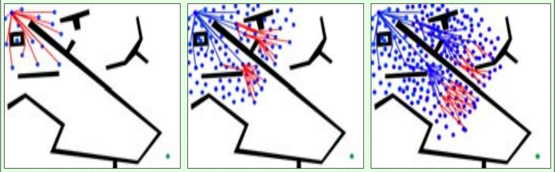
Parallelization of R* Search (R*GPU)

- 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 CPU, 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*GPU Pseudocode

```
enter into R*
generate n random succs from start state
Loop until goal is reached
  retrieve minimum n searches from heap
  send searches to GPU
  generate random succs for all searches
  retrieve search costs from GPU
  if search succeeds
    add succs from search to heap
if goal has been reached
  retrieve high level path from start to goal
  send search from state n to state n+1 for all states in high level path to GPU
  retrieve path between high level states from GPU
  combine into one path
```

R*GPU Planning



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

2D Planning

Obstacle Density	Performance Measure	Planner	(Simple)	(Hard)
20%	Best Cost	R*GPU	322.18	**310.15**
		R*	**316.75**	316.20
	Succ R*	R*GPU	69.87	**6.9**
		R*	**2461.55**	4.1
40%	Best Cost	R*GPU	**347.72**	**328.23**
		R*	349.90	348.89
	Succ R*	R*GPU	23.56	**4.21**
		R*	**45.54**	2.15
60%	Best Cost	R*GPU	**447.57**	n/a
		R*	499.60	n/a
	Succ R*	R*GPU	**5.94**	n/a
		R*	1.5	n/a

6 DOF Robot Arm

Performance Measure	ε	R*GPU/R*
Best Cost	2	0.965
	4	0.921
	6	0.918
# of Succ R*	2	38.5556
	4	37.516
	6	24.917
# of Local A*	2	24.899
	4	44.268
	6	64.262

Related Work

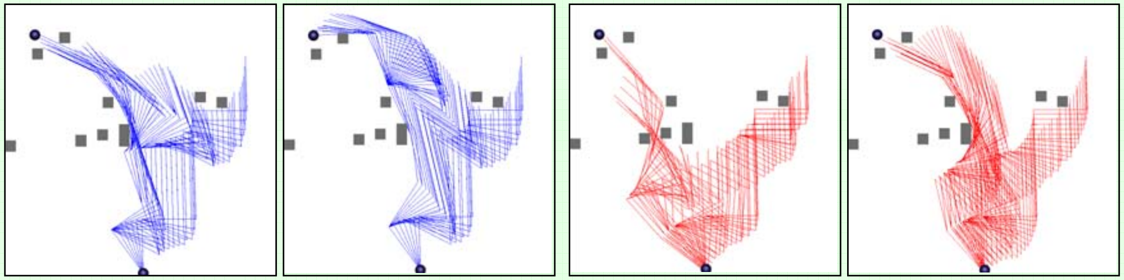
GPU Accelerated Path Finding [Bleiweiss 08]
- Optimally Navigate Agents
- Planning for 2D environments
- Exploits explicit Parallelism of multi-agent navigation planning
- A* Search kernals for many agents

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

A* Search [Nilsson 71] – A* Search

Experimental Results on a Simulated Robotic Arm

6 DOF Robotic Arm



GPU Cost = 78 | **GPU Cost = 81** | **CPU Cost = 89** | **CPU Cost = 101**

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