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

1 Introduction

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.

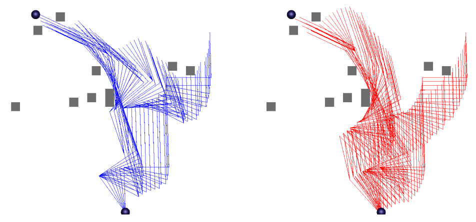
2 The Algorithm

R* search: R* search operates by decomposing the usual single-shot A* search [Nilsson 1971] into a series of properly-scheduled short-range and easy-to-solve searches, each guided by the heuristic function towards a randomly chosen goal. More specifically, R* constructs a small graph Γ of sparsely placed states, connected to each other via edges. Each edge in Γ represents a path in the original graph in between the corresponding states in the original graph.

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. To do this, R* grows Γ in the same way A* grows a search tree [Bleiweiss 2008]. At every iteration iteration, R* selects the next state s to expand from Γ. However, 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.

A path that R* returns is a path in Γ from the start state to the goal state. This path consists of edges in Γ. Each such edge, however, is a path in the original graph. Finding each of these (local) paths may potentially be a challenging planning task. 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 $\epsilon > 1$ to compute these easy-to-solve paths.

Parallelization of R* search: 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. In particular, 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. This results in significant speedups for the following reasons. First, each short-range search is independent of others which makes it suitable for running them in parallel. Second, each search is a short-range and easy-to-solve. This means that each search does not require vast amounts of memory. This allows for multiple searches to share states in the DRAM on the GPU so there are no unnecessary expansions. Finally, each search in R* discards its memory after it exits. This eliminates the need for time consuming transfers of memory and makes it ideal for running in the DRAM on the GPU.



(a) motion generated by R*GPU (cost=78)   (b) motion generated by R* (cost=101)

Figure 1: *Motions generated for a simulated 6 DOF robot arm after 30 secs of planning*

3 Results

We first evaluate the performance of R*GPU on 53 randomly generated 2D gridworlds of varying obstacle density and for fast (simple) and artificially time-consuming (hard) edgecost computations (Table 1(a)). The results show that R*GPU outperforms the CPU version of R* as obstacle density grows and as cost computation becomes time-consuming, which is often the case when planning for complex systems.

We also evaluate and compare the performance of R*GPU with the CPU version of R* on a simulated 6 degree of freedom (DOF) robotic arm (Figure 1) [Likhachev and Stentz 2008]. The base of the arm is fixed, and the task is to move its endeffector to the goal (small circle on the left) while navigating around obstacles (indicated by grey rectangles). The resulting state-space is over 3 billion states. The cost of each change in a joint angle is 1. We test our algorithm using three settings of ϵ. The results show that R*GPU consistently produces a lower "best cost", and within five minutes we can execute over 38 times more of successful R*GPU searches and over 64 times more of weighted A* searches than if we choose to run normal R* searches (Table 1 (b), note that shown numbers are ratios).

2D Planning				
Obstacle Density	Performance Measure	Planner	(Simple)	(Hard)
20%	Best Cost	R*GPU	322.18	**310.15**
		R*	**316.75**	316.20
	# of 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
	# of 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
	# of succ R*	R*GPU	**5.94**	n/a
		R*	1.5	n/a

(a) 2D planning environment results

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

(b) 6 DOF Robot Arm results

Table 1: *Experimental results. Performance Measures: Best Cost - the cost of the best solution found, # of Succ R* - the number of successful R* searches, # of Local A* - the number of short-range weighted A* searches executed within time allocated for planning*

References

BLEIWEISS, A. 2008. Gpu accelerated pathfinding. In *GH '08: Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 65–74.

LIKHACHEV, M., AND STENTZ, A. 2008. R* search. In *AAAI*, 344–350.

NILSSON, N. 1971. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill.