Donald Whyte
sc10dw

Accelerating Multi-Dimensional Search

Iteration #1 Update Meeting (17/03/14 to 24/03/14)

Summary of Progress So Far

Objective	**Progress**
Implement evaluation framework	Completed
Implement evaluation baselines (Sequential Scan and Octree)	Completed
Optimise evaluation baselines	*Not Complete*
Implement Splay Quadtree	Not Complete
Analyse performance of Splay Quadtree	Not Complete
Evaluate found performance of Splay Quadtree, comparing with baselines	Not Complete
Pyramid tree implemented	***Complete***
Findings	
Given Pyramid tree implementation had some inefficiencies that were fixed when it was decoupled from VTK • all vectors passed by value (many, many copies) • needless assignments of method parameters to local variables, which resulted in more copies • unnecessary for-loops • added remove() and update() operations	
As expected, the pyramid tree greatly outperforms sequential scan and octree implementations. For 10,000 operations on astrophysics dataset: • Pyramid tree: 0.64 seconds (reduced from ~3 seconds with copies) • Octree: 136.47 seconds	
Bottleneck in octree appears to be the remove() operation – took over 120 seconds of overall runtime!	
Bottleneck in Pyramid tree appears to be point equality checks and subscript access (due to non-contiguous memory). remove() can also cause problems due to how it handles memory de-allocation	

Donald Whyte
sc10dw

Objectives for This Iteration

1. ~~Implement evaluation framework~~
2. ~~Implement evaluation baselines (Sequential Scan and Quadtree)~~
3. *Optimise evaluation baselines (including Pyramid tree)*
4. Implement Splay Quadtree
5. Analyse performance of Splay Quadtree
6. Evaluate found performance of Splay Quadtree, comparing with baselines

New Items to Discuss

- Would like to spend some time optimising the two baselines and the Pyramid tree
  - means there most likely won't be enough time to complete the implementation of the Splay quadtree and evaluate it in this iteration
- Discussion of Pyramid tree implementation details
- Possible optimisations/ideas to explore for Pyramid tree
  - implement one-dimensional Splay tree for underlying 1D structure
    - still exploring self-adjusting behaviour with respect to high-dimensional data
  - instead of incrementally removing marked empty slots of point array, rebuilt entire structure at once
  - change indices in hash table buckets to the actual points (prevents removal issues)
- Trace of dynamic operations and queries (and the points used for said queries) from real application
  - Joint Contour Net construction
  - or some other application
- Content to put in end-of-iteration performance evaluation
  - Splay quadtree
  - Pyramid tree (excessive copies, things that were changed to increase performance)
  - Timings, remarks on profiling results (what methods/operations took the longest and why)
  - What small changes I made to increase performance (and how much increase it provided), as well as *why* the changes had such an impact on performance
  - Evaluate performance of baselines for this initial iteration as well?

Donald Whyte
sc10dw

Task Breakdown

Task	For Objective(s)	Start	End
Implement pooled, contiguous memory manager that is used to create point instances (and use for data generators/loaders)	*3*	*???*	*???*
Implement no-indices version of Pyramid tree	*3*	*???*	*???*
Implement rebuilding version of Pyramid tree	*3*	*???*	*???*
Implement Splay quadtree	4	18/03/14	20/03/14
Run evaluation framework on Splay Quadtree with various sets of test data	5	21/03/14	21/03/14
Produce short report (two pages max) containing performance evaluation of baselines	6	21/03/14	24/03/14