libkdtree

0.4.2

Generated by Doxygen 1.5.6

Sat Jul 31 23:13:58 2010

Contents

1 libkdtree - A C99 implementation of the kd-tree algorithm

Author:

Jörg Dietrich <jdietric@eso.org>

1.1 Introduction

libkdtree provides a C99 implementation of kd-trees. Kd-trees are space-partitioning trees that facilitate fast nearest-neighbor and range searches. The present implementation contains two flavors of kd-trees. The first kind of tree is constructed in n-dimensional Euclidean space; the second kind of tree is constructed from data on the 2-d surface of a sphere.

Other features are:

- Fast parallelized tree construction using POSIX threads
- Nearest neighbor search and
- N-nearest neighbor search
- Range search inside a (hyper-)sphere
- Range search inside a (hyper-)rectangle
- Sorted result lists are stored in an efficient min-max heap
- Storage of arbitrary data associated with each point

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

3 File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

4 Data Structure Documentation

4.1 kd_thread_data Struct Reference

arguments passed to the threaded kd-Tree construction

4.1.1 Detailed Description

arguments passed to the threaded kd-Tree construction

The documentation for this struct was generated from the following file:

- lib/kdtree.h

4.2 kdNode Struct Reference

kd-tree node structure definition

Data Fields

- float * location
- float * min
- float * max
- int split
- void * data
- struct kdNode * left
- struct kdNode * right

4.2.1 Detailed Description

kd-tree node structure definition

4.2.2 Field Documentation

4.2.2.1 float* kdNode::location

vector to the node's location

4.2.2.2 float* kdNode::min

vector to the min coordinates of the hyperrectangle

4.2.2.3 float* kdNode::max

vector to the max coordinates of the hyperrectangle

4.2.2.4 int kdNode::split

axis along which the tree bifurcates

4.2.2.5 void* kdNode::data

pointer to an optional data structure

4.2.2.6 struct kdNode* kdNode::left [read]

the left child of the tree node

4.2.2.7 struct kdNode* kdNode::right [read]

the right child of the tree node

The documentation for this struct was generated from the following file:

- lib/kdtree.h

4.3 pqueue Struct Reference

priority queue (min-max heap)

Data Fields

- uint32_t size
- uint32_t avail
- uint32_t step
- struct resItem ** d

4.3.1 Detailed Description

priority queue (min-max heap)

4.3.2 Field Documentation

4.3.2.1 uint32_t pqueue::size

current length of the queue

4.3.2.2 uint32_t pqueue::avail

currently allocated queue elements

4.3.2.3 uint32_t pqueue::step

step size in which new elements are allocated

4.3.2.4 struct resItem** pqueue::d [read]

pointer to an array of result items

The documentation for this struct was generated from the following file:

- lib/kdtree.h

4.4 resItem Struct Reference

result items, member of a priority queue

Data Fields

- struct kdNode * node
- float dist_sq

4.4.1 Detailed Description

result items, member of a priority queue

4.4.2 Field Documentation

4.4.2.1 struct kdNode* resItem::node [read]

pointer to a kdNode

4.4.2.2 float resItem::dist_sq

distance squared as the priority

The documentation for this struct was generated from the following file:

- lib/kdtree.h

5 File Documentation

5.1 lib/kdtree.h File Reference

Include file for the kdtree library.

Data Structures

- struct kdNode
  kd-tree node structure definition
- struct resItem
  result items, member of a priority queue
- struct pqueue
  priority queue (min-max heap)
- struct kd_thread_data
  arguments passed to the threaded kd-Tree construction

Functions

- struct pqueue * pqinit (struct pqueue *q, uint32_t n)
  Initialize priority queue.
- int pqinsert (struct pqueue *q, struct resItem *d)
  Insert an item into the queue.
- struct resItem ** pqremove_min (struct pqueue *q, struct resItem **d)
  remove the highest-ranking (minimum) item from the queue.
- struct resItem ** pqremove_max (struct pqueue *q, struct resItem **d)
  remove the lowest-ranking (maximum) item from the queue.
- struct resItem ** pqpeek_min (struct pqueue *q, struct resItem **d)
  access highest-ranking (minimum) item without removing it.
- struct resItem ** pqpeek_max (struct pqueue *q, struct resItem **d)
  access lowest-ranking (maximum) item without removing it.
- float kd_sph_xtd (float *p1, float *p2, float *p3)
  Compute cross-track distance.
- void kd_destroyTree (struct kdNode *node, void(*destr)(void *))
  free the kd-tree data structure,
- struct kdNode * kd_buildTree (struct kd_point *points, unsigned long nPoints, void *(*constr)(void *), void(*destr)(void *), float *min, float *max, int dim, int max_threads)

build kd-tree structure

- struct kdNode * kd_sph_buildTree (struct kd_point *points, unsigned long nPoints, void *(*constr)(void *), void(*destr)(void *), float *min, float *max, int max_threads)

  build kd-tree structure

- struct pqueue * kd_ortRangeSearch (struct kdNode *node, float *min, float *max, int dim)

  Perform orthogonal range search (get all points in a hyperrectangle).

- struct kdNode * kd_nearest (struct kdNode *node, float *p, float *max_dist_sq, int dim)

  Find the nearest neighbor of a point.

- struct pqueue * kd_qnearest (struct kdNode *node, float *p, float *max_dist_sq, unsigned int q, int dim)

  Return the q nearest-neighbors to a point.

- struct pqueue * kd_range (struct kdNode *node, float *p, float *max_dist_sq, int dim, int ordered)

  Perform a range search around a point.

- struct pqueue * kd_sph_ortRangeSearch (struct kdNode *node, float *min, float *max)

  Perform orthogonal range search (get all points in a hyperrectangle).

- struct kdNode * kd_sph_nearest (struct kdNode *node, float *p, float *max_dist_sq)

  Find the nearest neighbor of a point.

- struct pqueue * kd_sph_qnearest (struct kdNode *node, float *p, float *max_dist_sq, unsigned int q)

  Return the q nearest-neighbors to a point.

- struct pqueue * kd_sph_range (struct kdNode *node, float *p, float *max_dist_sq, int ordered)

  Perform a range search around a point.

5.1.1 Detailed Description

Include file for the kdtree library.

5.1.2 Function Documentation

5.1.2.1 struct kdNode* kd_buildTree (struct kd_point * *points*, unsigned long *nPoints*, void *(*)(void *) *constr*, void(*)(void *) *destr*, float * *min*, float * *max*, int *dim*, int *max_threads*) [read]

build kd-tree structure

Parameters:

points an array of kd_points (struct with position vector and data container).

nPoints the length of the points array.

constr a pointer to a void *constructor() function to include the data container in the tree; optional, can be NULL

destr a pointer to a void destructor() function to free() the data containers in the tree; optional, can be NULL, but should be given if the constr argument is non-NULL.

min a vector with the minimum positions of the corners of the hyperrectangle containing the data.

max a vector with the maximum positions of the corners of the hyperrectangle containing the data.

dim the dimensionality of the data.

max_threads the maximal number of threads spawned for construction of the tree. The threads will be unbalanced if this is not a power of 2.

Returns:

root node of the tree

5.1.2.2 void kd_destroyTree (struct kdNode * *node*, void(*)(void *) *destr*)

free the kd-tree data structure,

Parameters:

node the root node of the tree to be destroyed

**destr* a pointer to the destructor function for the data container.

Returns:

This function does not return a value

5.1.2.3 struct kdNode* kd_nearest (struct kdNode * *node*, float * *p*, float * *max_dist_sq*, int *dim*) [read]

Find the nearest neighbor of a point.

Parameters:

node the root node of the tree to be searched.

p a vector to the point whose nearest neighbor is sought.

max_dist_sq the square of the maximum distance to the nearest neighbor.

dim the dimension of the data.

Returns:

A pointer to node containing the nearest neighbor. max_dist_sq is set to the square of the distance to the nearest neigbor.

5.1.2.4 struct pqueue* kd_ortRangeSearch (struct kdNode * *node*, float * *min*, float * *max*, int *dim*) [read]

Perform orthogonal range search (get all points in a hyperrectangle).

Parameters:

node the root node of tree to be searched.

min a vector with the minimum positions of the corners of the hyperrectangle containing the data.

max a vector with the maximum positions of the corners of the hyperrectangle containing the data.

dim the dimension of the data.

Returns:

Pointer to a priority queue, NULL in case of problems.

5.1.2.5 struct pqueue* kd_qnearest (struct kdNode * *node*, float * *p*, float * *max_dist_sq*, unsigned int *q*, int *dim*) [read]

Return the q nearest-neighbors to a point.

Parameters:

node the root node of the tree to be searched.

p a vector to the point whose nearest neighbors are sought.

max_dist_sq the square of the maximum distance to the nearest neighbors.

q the maximum number of points to be retured.

dim the dimension of the data.

Returns:

A pointer to a priority queue of the points found, or NULL in case of problems.

5.1.2.6 struct pqueue* kd_range (struct kdNode * *node*, float * *p*, float * *max_dist_sq*, int *dim*, int *ordered*) [read]

Perform a range search around a point.

Parameters:

node the root node of the tree to be searched.

p the location of the point around which the search is carried out .

max_dist_sq the square of the radius of the hypersphere.

dim the dimension of the data.

ordered determines whether the result list should be ordered in increasing distance (KD_ORDERED) or unordered (KD_UNORDERED).

Returns:

A pointer to a priority queue containing the points found, NULL in case of problems.

5.1.2.7 struct kdNode* kd_sph_buildTree (struct kd_point * *points*, unsigned long *nPoints*, void *(*)(void *) *constr*, void(*)(void *) *destr*, float * *min*, float * *max*, int *max_threads*) [read]

build kd-tree structure

Parameters:

points an array of kd_points (struct with position vector and data container).

nPoints the length of the points array.

constr a pointer to a void ∗constructor() function to include data container in tree; optional, can be NULL

destr a pointer to a void destructor() function to free() data container in the tree; optional, can be NULL, but should be given if the constr argument is non-NULL.

min a vector with the minimum positions of the corners of the hyperrectangle containing the data.

max a vector with the maximum positions of the corners of the hyperrectangle containing the data.

max_threads the maximal number of threads spawned for construction of the tree. The threads will be unbalanced if this is not a power of 2.

Returns:

root node of the tree

5.1.2.8 struct kdNode∗ kd_sph_nearest (struct kdNode ∗ *node*, float ∗ *p*, float ∗ *max_dist_sq*) [read]

Find the nearest neighbor of a point.

Parameters:

node the root node of the tree to be searched.

p a vector to the point whose nearest neighbor is sought.

max_dist_sq the square of the maximum distance to the nearest neighbor.

Returns:

A pointer to node containing the nearest neighbor. max_dist_sq is set to the square of the distance to the nearest neigbor.

5.1.2.9 struct pqueue∗ kd_sph_ortRangeSearch (struct kdNode ∗ *node*, float ∗ *min*, float ∗ *max*) [read]

Perform orthogonal range search (get all points in a hyperrectangle).

Parameters:

node the root node of tree to be searched.

min a vector with the minimum positions of the corners of the hyperrectangle containing the data.

max a vector with the maximum positions of the corners of the hyperrectangle containing the data.

Returns:

Pointer to a priority queue, NULL in case of problems.

Rectangle must not cross the meridian!

5.1.2.10 struct pqueue∗ kd_sph_qnearest (struct kdNode ∗ *node*, float ∗ *p*, float ∗ *max_dist_sq*, unsigned int *q*) [read]

Return the q nearest-neighbors to a point.

Parameters:

node the root node of the tree to be searched.

p a vector to the point whose nearest neighbors are sought.

max_dist_sq the square of the maximum distance to the nearest neighbors.

q the maximum number of points to be retured.

Returns:

A pointer to a priority queue of the points found, or NULL in case of problems.

5.1.2.11 struct pqueue* kd_sph_range (struct kdNode * *node*, float * *p*, float * *max_dist_sq*, int *ordered*) [read]

Perform a range search around a point.

Parameters:

node the root node of the tree to be searched.

p the location of the point around which the search is carried out .

max_dist_sq the square of the radius of the hypersphere.

ordered determines whether the result list should be ordered in increasing distance (KD_ORDERED) or unordered (KD_UNORDERED).

Returns:

A pointer to a priority queue containing the points found, NULL in case of problems.

5.1.2.12 float kd_sph_xtd (float * *p1*, float * *p2*, float * *p3*)

Compute cross-track distance.

Parameters:

p1 a point

p2 a second point defining a great circle from p1

p3 a third point whose distance from the great circle we compute

Returns:

distance of p3 from the great circle connecting p1 and p2.

5.1.2.13 struct pqueue* pqinit (struct pqueue * *q*, uint32_t *n*) [read]

Initialize priority queue.

Parameters:

q a pointer to a priority queue, or NULL if the queue should be initialized.

n the number of queue items for which memory should be preallocated. If you insert more than n items to the queue, another n items will be allocated automatically.

Returns:

Pointer to priority queue, NULL in case of error.

5.1.2.14 int pqinsert (struct pqueue * *q*, struct resItem * *d*)

Insert an item into the queue.

Parameters:

q a pointer to a priority queue.

d the datum to be inserted.

Returns:

1 if the item has been inserted, 0 if the item could not be appended. Either the queue pointer provided was NULL, or the function was unable to allocate the amount of memory needed for the new item.

5.1.2.15 struct resItem** pqpeek_max (struct pqueue * *q*, struct resItem ** *d*) [read]

access lowest-ranking (maximum) item without removing it.

Parameters:

q a pointer to a priority queue.

d a pointer to the struct resItem * variable that will hold the datum corresponding to the highest-ranking item.

Returns:

non-NULL in case of success. The variable that d points to now contains the datum associated with the highest-ranking item; NULL in case of failure. Either the queue pointer provided was NULL, or the queue was empty. The chunk of memory that d points to has not been modified.

5.1.2.16 struct resItem** pqpeek_min (struct pqueue * *q*, struct resItem ** *d*) [read]

access highest-ranking (minimum) item without removing it.

Parameters:

q a pointer to a priority queue.

d a pointer to the struct resItem * variable that will hold the datum corresponding to the highest-ranking item.

Returns:

non-NULL in case of success. The variable that d points to now contains the datum associated with the highest-ranking item; NULL in case of failure. Either the queue pointer provided was NULL, or the queue was empty. The chunk of memory that d points to has not been modified.

5.1.2.17 struct resItem** pqremove_max (struct pqueue * *q*, struct resItem ** *d*) [read]

remove the lowest-ranking (maximum) item from the queue.

Parameters:

q a pointer to a priority queue.

d a pointer to the struct resItem * variable that will hold the datum corresponding to the queue item removed.

Returns:

non-NULL if an item has been removed. The variable that d points to now contains the datum associated with the item in question; or NULL if item could be removed. Either the queue pointer provided was NULL, or the queue was empty. The chunk of memory that d points to has not been modified.

5.1.2.18 struct resItem** pqremove_min (struct pqueue * *q*, struct resItem ** *d*) [read]

remove the highest-ranking (minimum) item from the queue.

Parameters:

q a pointer to a priority queue.

d a pointer to the struct resItem * variable that will hold the datum corresponding to the queue item removed.

Returns:

non-NULL if an item has been removed. The variable that d points to now contains the datum associated with the item in question; or NULL if item could be removed. Either the queue pointer provided was NULL, or the queue was empty. The chunk of memory that d points to has not been modified.

5.2 lib/kdtree_cartesian.c File Reference

Routines specific to the n-dimensional cartesian kd-tree.

Functions

- struct kdNode * kd_buildTree (struct kd_point *points, unsigned long nPoints, void *(*constr)(void *), void(*destr)(void *), float *min, float *max, int dim, int max_threads)

  build kd-tree structure

- struct pqueue * kd_ortRangeSearch (struct kdNode *node, float *min, float *max, int dim)

  Perform orthogonal range search (get all points in a hyperrectangle).

- struct kdNode * kd_nearest (struct kdNode *node, float *p, float *max_dist_sq, int dim)

  Find the nearest neighbor of a point.

- struct pqueue * kd_qnearest (struct kdNode *node, float *p, float *max_dist_sq, unsigned int q, int dim)

  Return the q nearest-neighbors to a point.

- struct pqueue * kd_range (struct kdNode *node, float *p, float *max_dist_sq, int dim, int ordered)

  Perform a range search around a point.

5.2.1 Detailed Description

Routines specific to the n-dimensional cartesian kd-tree.

5.2.2 Function Documentation

5.2.2.1 struct kdNode* kd_buildTree (struct kd_point * *points*, unsigned long *nPoints*, void *(*)(void *) *constr*, void(*)(void *) *destr*, float * *min*, float * *max*, int *dim*, int *max_threads*) [read]

build kd-tree structure

Parameters:

points an array of kd_points (struct with position vector and data container).

nPoints the length of the points array.

constr a pointer to a void *constructor() function to include the data container in the tree; optional, can be NULL

destr a pointer to a void destructor() function to free() the data containers in the tree; optional, can be NULL, but should be given if the constr argument is non-NULL.

min a vector with the minimum positions of the corners of the hyperrectangle containing the data.

max a vector with the maximum positions of the corners of the hyperrectangle containing the data.

dim the dimensionality of the data.

max_threads the maximal number of threads spawned for construction of the tree. The threads will be unbalanced if this is not a power of 2.

Returns:

root node of the tree

5.2.2.2 struct kdNode* kd_nearest (struct kdNode * *node*, float * *p*, float * *max_dist_sq*, int *dim*) [read]

Find the nearest neighbor of a point.

Parameters:

node the root node of the tree to be searched.

p a vector to the point whose nearest neighbor is sought.

max_dist_sq the square of the maximum distance to the nearest neighbor.

dim the dimension of the data.

Returns:

A pointer to node containing the nearest neighbor. max_dist_sq is set to the square of the distance to the nearest neigbor.

5.2.2.3 struct pqueue* kd_ortRangeSearch (struct kdNode * *node*, float * *min*, float * *max*, int *dim*) [read]

Perform orthogonal range search (get all points in a hyperrectangle).

Parameters:

node the root node of tree to be searched.

min a vector with the minimum positions of the corners of the hyperrectangle containing the data.

max a vector with the maximum positions of the corners of the hyperrectangle containing the data.

dim the dimension of the data.

Returns:

Pointer to a priority queue, NULL in case of problems.

5.2.2.4 struct pqueue* kd_qnearest (struct kdNode * *node*, float * *p*, float * *max_dist_sq*, unsigned int *q*, int *dim*) [read]

Return the q nearest-neighbors to a point.

Parameters:

node the root node of the tree to be searched.

p a vector to the point whose nearest neighbors are sought.

max_dist_sq the square of the maximum distance to the nearest neighbors.

q the maximum number of points to be retured.

dim the dimension of the data.

Returns:

A pointer to a priority queue of the points found, or NULL in case of problems.

5.2.2.5 struct pqueue* kd_range (struct kdNode * *node*, float * *p*, float * *max_dist_sq*, int *dim*, int *ordered*) [read]

Perform a range search around a point.

Parameters:

node the root node of the tree to be searched.

p the location of the point around which the search is carried out .

max_dist_sq the square of the radius of the hypersphere.

dim the dimension of the data.

ordered determines whether the result list should be ordered in increasing distance (KD_ORDERED) or unordered (KD_UNORDERED).

Returns:

A pointer to a priority queue containing the points found, NULL in case of problems.

5.3 lib/kdtree_common.c File Reference

Routines common to the Cartesian and spherical kd-tree versions.

Functions

- void kd_destroyTree (struct kdNode *node, void(*destr)(void *))

  free the kd-tree data structure,

5.3.1 Detailed Description

Routines common to the Cartesian and spherical kd-tree versions.

5.3.2 Function Documentation

5.3.2.1 void kd_destroyTree (struct kdNode * *node*, void(*)(void *) *destr*)

free the kd-tree data structure,

Parameters:

node the root node of the tree to be destroyed

destr a pointer to the destructor function for the data container.

Returns:

This function does not return a value

5.4 lib/kdtree_spherical.c File Reference

Routines specific to the 2-dimensional spherical kd-tree.

Functions

- float kd_sph_xtd (float *p1, float *p2, float *p3)
  Compute cross-track distance.
- struct kdNode * kd_sph_buildTree (struct kd_point *points, unsigned long nPoints, void *(*constr)(void *), void(*destr)(void *), float *min, float *max, int max_threads)
  build kd-tree structure
- struct pqueue * kd_sph_ortRangeSearch (struct kdNode *node, float *min, float *max)
  Perform orthogonal range search (get all points in a hyperrectangle).
- struct kdNode * kd_sph_nearest (struct kdNode *node, float *p, float *max_dist_sq)
  Find the nearest neighbor of a point.
- struct pqueue * kd_sph_qnearest (struct kdNode *node, float *p, float *max_dist_sq, unsigned int q)
  Return the q nearest-neighbors to a point.
- struct pqueue * kd_sph_range (struct kdNode *node, float *p, float *max_dist_sq, int ordered)
  Perform a range search around a point.

5.4.1 Detailed Description

Routines specific to the 2-dimensional spherical kd-tree.

5.4.2 Function Documentation

5.4.2.1 struct kdNode* kd_sph_buildTree (struct kd_point * *points*, unsigned long *nPoints*, void *(*)(void *) *constr*, void(*)(void *) *destr*, float * *min*, float * *max*, int *max_threads*) [read]

build kd-tree structure

Parameters:

points an array of kd_points (struct with position vector and data container).

nPoints the length of the points array.

constr a pointer to a void *constructor() function to include data container in tree; optional, can be NULL

destr a pointer to a void destructor() function to free() data container in the tree; optional, can be NULL, but should be given if the constr argument is non-NULL.

min a vector with the minimum positions of the corners of the hyperrectangle containing the data.

max a vector with the maximum positions of the corners of the hyperrectangle containing the data.

max_threads the maximal number of threads spawned for construction of the tree. The threads will be unbalanced if this is not a power of 2.

Returns:

root node of the tree

5.4.2.2 struct kdNode* kd_sph_nearest (struct kdNode * *node*, float * *p*, float * *max_dist_sq*) [read]

Find the nearest neighbor of a point.

Parameters:

node the root node of the tree to be searched.

p a vector to the point whose nearest neighbor is sought.

max_dist_sq the square of the maximum distance to the nearest neighbor.

Returns:

A pointer to node containing the nearest neighbor. max_dist_sq is set to the square of the distance to the nearest neigbor.

5.4.2.3 struct pqueue* kd_sph_ortRangeSearch (struct kdNode * *node*, float * *min*, float * *max*) [read]

Perform orthogonal range search (get all points in a hyperrectangle).

Parameters:

node the root node of tree to be searched.

min a vector with the minimum positions of the corners of the hyperrectangle containing the data.

max a vector with the maximum positions of the corners of the hyperrectangle containing the data.

Returns:

Pointer to a priority queue, NULL in case of problems.

Rectangle must not cross the meridian!

5.4.2.4 struct pqueue* kd_sph_qnearest (struct kdNode * *node*, float * *p*, float * *max_dist_sq*, unsigned int *q*) [read]

Return the q nearest-neighbors to a point.

Parameters:

node the root node of the tree to be searched.

p a vector to the point whose nearest neighbors are sought.

max_dist_sq the square of the maximum distance to the nearest neighbors.

q the maximum number of points to be retured.

Returns:

A pointer to a priority queue of the points found, or NULL in case of problems.

5.4.2.5 struct pqueue* kd_sph_range (struct kdNode * *node*, float * *p*, float * *max_dist_sq*, int *ordered*) [read]

Perform a range search around a point.

Parameters:

node the root node of the tree to be searched.

p the location of the point around which the search is carried out .

max_dist_sq the square of the radius of the hypersphere.

ordered determines whether the result list should be ordered in increasing distance (KD_ORDERED) or unordered (KD_UNORDERED).

Returns:

A pointer to a priority queue containing the points found, NULL in case of problems.

5.4.2.6 float kd_sph_xtd (float * *p1*, float * *p2*, float * *p3*)

Compute cross-track distance.

Parameters:

p1 a point

p2 a second point defining a great circle from p1

p3 a third point whose distance from the great circle we compute

Returns:

distance of p3 from the great circle connecting p1 and p2.

5.5 lib/pqueue.c File Reference

Two-ended priority queues (min-max heap).

Functions

- struct pqueue * pqinit (struct pqueue *q, uint32_t n)
  Initialize priority queue.
- int pqinsert (struct pqueue *q, struct resItem *d)
  Insert an item into the queue.
- struct resItem ** pqremove_min (struct pqueue *q, struct resItem **d)
  remove the highest-ranking (minimum) item from the queue.
- struct resItem ** pqremove_max (struct pqueue *q, struct resItem **d)
  remove the lowest-ranking (maximum) item from the queue.
- struct resItem ** pqpeek_min (struct pqueue *q, struct resItem **d)
  access highest-ranking (minimum) item without removing it.
- struct resItem ** pqpeek_max (struct pqueue *q, struct resItem **d)
  access lowest-ranking (maximum) item without removing it.

5.5.1 Detailed Description

Two-ended priority queues (min-max heap).

Implementation of a min-max heap (two-ended priority queue) as introduced by Atkinson et al. (1986), Communications of the ACM 10, 996.

5.5.2 Function Documentation

5.5.2.1 struct pqueue* pqinit (struct pqueue * *q*, uint32_t *n*) [read]

Initialize priority queue.

Parameters:

q a pointer to a priority queue, or NULL if the queue should be initialized.

n the number of queue items for which memory should be preallocated. If you insert more than n items to the queue, another n items will be allocated automatically.

Returns:

Pointer to priority queue, NULL in case of error.

5.5.2.2 int pqinsert (struct pqueue * *q*, struct resItem * *d*)

Insert an item into the queue.

Parameters:

q a pointer to a priority queue.

d the datum to be inserted.

Returns:

1 if the item has been inserted, 0 if the item could not be appended. Either the queue pointer provided was NULL, or the function was unable to allocate the amount of memory needed for the new item.

5.5.2.3 struct resItem** pqpeek_max (struct pqueue * *q*, struct resItem ** *d*) [read]

access lowest-ranking (maximum) item without removing it.

Parameters:

q a pointer to a priority queue.

d a pointer to the struct resItem * variable that will hold the datum corresponding to the highest-ranking item.

Returns:

non-NULL in case of success. The variable that d points to now contains the datum associated with the highest-ranking item; NULL in case of failure. Either the queue pointer provided was NULL, or the queue was empty. The chunk of memory that d points to has not been modified.

5.5.2.4 struct resItem** pqpeek_min (struct pqueue * *q*, struct resItem ** *d*) [read]

access highest-ranking (minimum) item without removing it.

Parameters:

q a pointer to a priority queue.

d a pointer to the struct resItem * variable that will hold the datum corresponding to the highest-ranking item.

Returns:

non-NULL in case of success. The variable that d points to now contains the datum associated with the highest-ranking item; NULL in case of failure. Either the queue pointer provided was NULL, or the queue was empty. The chunk of memory that d points to has not been modified.

5.5.2.5 struct resItem** pqremove_max (struct pqueue * *q*, struct resItem ** *d*) [read]

remove the lowest-ranking (maximum) item from the queue.

Parameters:

q a pointer to a priority queue.

d a pointer to the struct resItem * variable that will hold the datum corresponding to the queue item removed.

Returns:

non-NULL if an item has been removed. The variable that d points to now contains the datum associated with the item in question; or NULL if item could be removed. Either the queue pointer provided was NULL, or the queue was empty. The chunk of memory that d points to has not been modified.

5.5.2.6 struct resItem** pqremove_min (struct pqueue * *q*, struct resItem ** *d*) [read]

remove the highest-ranking (minimum) item from the queue.

Parameters:

q a pointer to a priority queue.

d a pointer to the struct resItem * variable that will hold the datum corresponding to the queue item removed.

Returns:

non-NULL if an item has been removed. The variable that d points to now contains the datum associated with the item in question; or NULL if item could be removed. Either the queue pointer provided was NULL, or the queue was empty. The chunk of memory that d points to has not been modified.

5.6 tests/test_cartesian_nn.c File Reference

compare kdTree NN search with naive NN.

5.6.1 Detailed Description

compare kdTree NN search with naive NN.

5.7 tests/test_spherical_nn.c File Reference

compare kdTree NN search with naive NN.

5.7.1 Detailed Description

compare kdTree NN search with naive NN.

Index