Published in final edited form as:

Phys Med Biol. ; 64(15): 155018. doi:10.1088/1361-6560/ab0da6.

Fast gamma-ray interaction-position estimation using k-d tree search

Xin Li^{1,3}, Li Tao², Craig S Levin², Lars R Furenlid¹

¹Center for Gamma-Ray Imaging and College of Optical Sciences, University of Arizona, Tucson, AZ 85721, United States of America

²Molecular Imaging Instrumentation Laboratory, Stanford University, Stanford, CA 94305, United States of America

Abstract

We have developed a fast gamma-ray interaction-position estimation method using k-d tree search, which can be combined with various kinds of closeness metrics such as Euclidean distance, maximum-likelihood estimation, etc. Compared with traditional search strategies, this method can achieve both speed and accuracy at the same time using the k-d tree data structure. The k-d tree search method has a time complexity of $O(\log_2(N))$, where N is the number of entries in the reference data set, which means large reference datasets can be used to efficiently estimate each event's interaction position. This method's accuracy was found to be equal to that of the exhaustive search method, yielding the highest achievable accuracy. Most importantly, this method has no restriction on the data structure of the reference dataset and can still work with complicated mean-detector-response functions (MDRFs), meaning that it is more robust than other popular methods such as contracting-grid-search (CG) or vector-search (VS) methods that could yield locally optimal instead of globally optimal results.

Keywords

photon counting; detector; gamma ray; k-d tree; estimation

1. Introduction

Scintillator-based detectors are extensively used in both clinical and pre-clinical single-photon emission computed tomography (SPECT) and positron emission tomography (PET) scanners due to their relatively low cost, high gamma-ray stopping power, and fast timing (Lewellen 2008). Position-estimation algorithms or decoding methods are applied for gamma-ray interaction position localization with signals induced by scintillation photons impinging on light sensors such as photomultiplier tubes (PMTs), silicon photomultipliers (SiPMs), etc. Anger arithmetic has dominated in gamma-ray interaction position estimation for decades due to its simplicity and fast estimation/decoding speed (Anger 1958). However, driven by the demand for better spatial resolution and lower bias, many other gamma-ray

³Author to whom correspondence should be addressed. xli@optics.arizona.edu.

position estimation/decoding algorithms based on reference datasets have been proposed and studied, such as look-up tables (LUT) (Aarsvold *et al* 1995), contracting-grid searches (Hesterman *et al* 2010), k-nearest neighbor searches (van Dam *et al* 2011), neural networks (Yonggang *et al* 2011), etc... These reference-data based methods not only provide better positioning capability for gamma-ray interactions, especially at detector edges and corners, but can also filter out events below a lower merit threshold (for example, likelihood (Barrett *et al* 2009)) to reject events such as those that Compton scatter and deposit energy in multiple positions in the detector.

Unlike Anger arithmetic, reference-data-based methods need detector calibration procedures to acquire their reference datasets, and the position-estimation speed remains relatively slow due to the extensive computations required. The current need for position estimation is to achieve both accuracy and efficiency at the same time. Our team has been using the mean-detector-response functions (MDRFs) derived from calibration, to represent the characteristics of the detector, and search among these candidate positions in the MDRFs using the contracting-grid (CG) method to find the position with highest likelihood compared to other candidates (ML) (Hesterman *et al* 2010). However, this method may fail to find the global maximum with detectors having complicated MDRFs such as edge-readout detectors with optical barriers (Li *et al* 2018). In this work, we propose a method that can accurately estimate gamma-ray interaction positions with satisfactory accuracy and short search time. This method uses a multi-dimensional tree structure (k-d tree) (Bentley 1975) constructed with mean-detector-response functions (MDRFs) to assist finding the nearest neighbor of the gamma-ray event signal among the candidates in the MDRFs.

The k-d tree search method, in fact, offers superior accuracy since the position estimates are equivalent to those found with exhaustive search methods. Also, utilizing the k-d tree data structure requires much less computation, so the search time is reduced.

2. Materials and methods

2.1. Brief introduction about *k-d tree*

K-d tree stands for k-dimensional tree – a tree structure constructed with data of k dimensions. The basic idea is to use high dimensional hyper planes to subdivide the reference dataset into many partitions, which will eventually assist the searching process.

2.1.1. One-dimensional case—Suppose we have an array of eight numbers: {1, 2, 2.3, 4, 5.5, 7, 8.1, 8.2}, then a 1D tree can be constructed. The first node value splits the whole array of numbers into two parts. The array elements smaller than the node's value go to the left branch of the node, while the array elements greater than the node's value go to the right branch below the node. The second and third nodes can be found in the left and right branches of the root node, respectively, using the same procedure. The nodes are constructed continuously until only one element is left in a branch, and this element is defined to comprise a leaf. A 1D tree constructed is shown in figure 1, and all of the eight elements are stored as leaves.

The value of first node of the 1D tree was chosen to be 4.75, which is actually the mean of 4 and 5.5 for simplicity. The second node's value is the medium of the numbers in the left branch of the first node, that is the medium of {1, 2, 2.3, 4}, which is the mean of 2 and 2.3. This procedure is repeated until all the numbers in the array become leaves, indicating that the binary search tree has been constructed. The 1D tree can sort the numbers in either ascending or descending order, depending on the rule to construct the tree.

This 1D tree is also a binary search tree, and searching for an element can be very efficient, especially when the total number of leaves becomes large. For example, if we want to see if the number 4 is in the tree or not, we start with comparing 4 with the root node's value which is 4.75; since $4 < 4.75$, we start to search the node's left branch. Then we compare 4 with 2.15, and since $4 > 2.15$, we start to search the node's right branch until we find 4 in leaf node. The total time complexity of searching a binary tree is $O(\log_2(N))$, where N is the number of total leaves in the binary tree.

2.1.2. Two-dimensional case—A 2D tree is a two-dimensional binary search tree, whose leaves have two dimensions. Each node of the 2D tree also splits the current dataset into two data subsets. Since the dataset contains 2D vectors, it is vital to decide in which of the two dimensions to set the value that splits the dataset. In the following example, the splitting dimension is randomly picked.

The dataset (containing 2D vectors) to construct an example 2D tree is: {[0.3, 1.5], [2.3, 3], [2.1, 1.3], [3, 3], [1.7, 3.7], [2, 0.3], [3, 5], [2.6, 6], [3, 2], [3.1, 4], [1, 3], [2, 4], [5.2, 1], [4.8, 2], [4, 0.2], [4, 1], }. The corresponding constructed 2D tree is shown in figure 2.

The 16 2D vectors can be plot on a 2D map, as shown in figure 3.

The nodes of the 2D tree work as partitions to segment the region containing the 2D vectors (points in the map). For example, the first node $y: 2.5$ is a line ($y = 2.5$), separating 2D vectors equally into two groups. The 2D vectors whose y components are smaller than 2.5 are classified into its left branch, while the 2D vectors whose y components are larger than 2.5 are classified into its right branch. The 2D map segmented by the 2D tree is shown in figure 4.

After the construction of the 2D tree, a search can be performed similarly to the 1D case. For example, to search whether a 2D vector [1, 3] is in the tree or not, the y component value 3 of this vector is compared with 2.5; since $3 > 2.5$, the search should continue in the right branch. Then, the x component value 1 of this vector is compared with 2.3; since $1 < 2.3$, the subsequent search should be done in the left branch. Then 3 is compared with 3.3; since $3 < 3.3$, we continue to search its left branch. Then 1 is compared with 1.75, since $1 < 1.75$, and the left branch is a leaf, we compare the vector [1, 3] with this leaf and find out they are equal. Then we can conclude that the vector [1, 3] is in this 2D tree.

2.1.3. Multi-dimensional tree—A k -d tree is a binary tree based on data vectors with k dimensions. Similar to the 2D tree case, each node has two attributes. The first attribute is the index of dimension, the second attribute is the value indicating the boundary for

partition. The index of dimensionality indicates on which axis the partition occurs, and the partition value indicates the specific partition position on the axis.

A 3D tree can be viewed as series of planes segmenting the 3D space into multiple subspaces, that contains data points as equally as possible, as shown in figure 5. It is hard to visualize a 4D or 5D tree, but the basic principle is the same.

Note that the criteria for constructing each node of the multi-dimensional tree is that the dimension and value that ‘best’ split the reference data (or entry data) are chosen. Under the influence of noise, a small gap between two branches will cause crosstalk between its two branches, meaning that events could be misclassified into the wrong branch, which induces search error, and this would slow down the searching process as introduced below.

Therefore, the gaps between two branches should be as large as possible, especially for the nodes close to the root node.

2.2. Searching for k-nearest neighbor (KNN)

One application of the k-d tree is to search for the presence of a specific data event efficiently. However, other methods, such as hash tables or maps, can do that even more efficiently. Moreover, when the event does not exactly match any entry (leaf) of the k-d tree, one may still be interested in finding the unknown event’s nearest neighbors among the entries (leaves). That is, to use certain figure-of-merit metric to measure the distance between the event and all the entries in the k-d tree and find the k entries with the highest or lowest merit values. This problem is called the k-nearest neighbor problem. Note that the ‘k’ here is different from the ‘k’ in k-d tree: whereas the ‘k’ in ‘k-nearest neighbor’ represents the number of entries in a dataset with highest or lowest merit values, the ‘k’ in ‘k-d tree’ stands for the dimensionality of the events inside the tree. In this work, only the ‘nearest’ entry is searched for, which corresponds to a 1-nearest neighbor problem.

For example, if the query event is denoted as $\vec{e}$, one entry inside the k-d tree is noted as $\vec{l}$, the dimensionality is M , the L_p norm might be used as the merit value to measure their closeness:

$$\text{merit}(\vec{l}) = \left(\sum_{i=1}^M |e_i - l_i|^p \right)^{1/p}. \quad (1)$$

The goal is to find the entry in the reference dataset with the smallest distance to the query event. An exhaustive search can be used to solve this problem, but the relatively long searching time is a big disadvantage for processing many events. The k-d tree can be used to organize the entries in the reference dataset into a binary search tree structure across k dimensions. The search for the nearest neighbor for the query event can then be performed efficiently.

2.2.1. Pruning of a branch—The searching process comprises traversing the whole tree while following certain strategy. A depth-first search strategy is used in this algorithm. Whereas the branch that satisfies the partition criteria is always searched first. For example,

if at a node where $event[node.idx] < node.value$, then the left branch is searched first (we define the left branch as the one that all of its entries satisfy $entry[node.idx] < node.value$, where $node.idx$ indicates the dimension to split and $node.value$ is the boundary value). When a leaf node is visited, equation (1) can be used to calculate the distance between the event and the leaf; in this example the p in equation (1) is set as 2, which corresponds to a Euclidean distance for simplicity. The current minimum distance ($merit_{min}$) and its corresponding entry values (reference vector) are stored.

However, in order to accelerate the searching process, some branches can be pruned (not searched) if certain conditions are met. A vector $\overrightarrow{cost}$ is used to store the deviation in each dimension, so the vector $\overrightarrow{cost}$ has the same dimension as an event vector. The deviation is caused by choosing the branch that violates the partition criteria. For example, if at a node, $event[node.idx] < node.value$, but we search the node's right branch instead of the left branch, which violates the partition criteria. Then a deviation (dev) is defined as $|event[node.idx] - node.value|$. If $dev > cost[node.idx]$, then the deviation dev is used to update the corresponding entry of $\overrightarrow{cost}$ ($cost[node.idx] = dev$). Then before searching the right branch, the total deviation $dev_{total} = \left(\sum_{i=1}^M cost[i]^2\right)^{1/2}$ is computed. If $dev_{total} < merit_{min}$, the search can be extended to include the right branch; otherwise, there is no need to search this branch as the accumulated error already exceeds that of the current best result, meaning no better (closer) entry can be found in the right branch.

The pruning can drastically increase the search speed, since a large fraction of branches can be excluded from the searching path for a specific event. This is the key element of k-d tree search, since if no pruning is done, the total search time is equivalent to that of the exhaustive-search method.

2.2.2. A 2D tree search example—Consider the 2D tree in figure 2 as an example. If the query event is [4.8, 2.6], and we want to find its nearest neighbor in the tree, then the search process can be described as follows:

1. At root node y : 2.5, compare y components; since $2.6 > 2.5$, proceed to node x : 2.3.
2. Compare the x component; since $4.8 > 2.3$, proceed to node y : 4.45.
3. Compare y components; since $2.6 < 4.45$, proceed to node y : 3.5.
4. Compare y components; since $2.6 < 3.5$ and since the left branch is a leaf, compare the query event with the leaf. The distance between the query vector and this leaf is $\sqrt{(4.8 - 3)^2 + (2.6 - 3)^2} \approx 1.84$. Record this distance as $merit_{min}$; then we return to node y : 3.5.
5. Before visiting its right branch, the total deviation needs to be calculated and compared with $merit$. Because $dev_{total} = |3.5 - 2.6| = 0.9$ which is smaller than $merit_{min} = 1.84$, a closer entry could still be found in this node's right branch and we continue to search there.

6. Compare the query vector [4.8, 2.6] with the leaf [3.1, 4]; the distance is 2.20, which is larger than $merit_{min}$, then return to node y : 3.5.
7. Since both the branches of node y : 3.5 have been visited, return to node y : 4.45.
8. Since $dev_{total} = |4.45 - 2.6| = 1.95 > merit_{min} = 1.84$, there is no need to search its right branch; proceed to node x : 2.3.
9. Since $dev_{total} = |4.8 - 2.3| = 2.5 > merit_{min} = 1.84$, there is no need to search its left branch; return to node y : 2.5.
10. Since $dev_{total} = |2.6 - 2.5| = 0.1 < merit_{min} = 1.84$, it is possible to find a better result in its left branch, then go to node x : 3.4.
11. Compare x components; since $4.8 > 3.4$, proceed to node x : 4.4.
12. Compare x components; since $4.8 > 4.4$, proceed to node y : 1.4.
13. Compare y components; since $2.6 > 1.4$, and the right branch is a leaf, compare the query vector [4.8, 2.6] with the leaf [4.8, 2]. The distance is $0.6 < merit_{min} = 1.84$, thus a better result is found and we update $merit_{min}$ with 0.6 and record this leaf [4.8, 2]. Return to node y : 1.4.
14. Since $dev_{total} = |2.6 - 1.4| = 1.2 > merit_{min} = 0.6$, there is no need to search its left branch; return to node x : 4.4.
15. Since $dev_{total} = \sqrt{(4.8 - 4.4)^2 + (2.6 - 2.5)^2} \approx 0.412$ (considering cumulative error in both x and y dimensions, remember there is already 0.1 error in y dimension) $< merit_{min} = 0.6$, a better result might be found in the left branch; proceed to node y : 0.65.
16. Compare y components; since $2.6 > 0.65$, compare the query vector with its right leaf. The distance is $1.79 > merit_{min} = 0.6$, return to node y : 0.65.
17. Since $dev_{total} = \sqrt{(4.8 - 4.4)^2 + (2.6 - 0.65)^2} \approx 1.99 > merit_{min} = 0.6$, there is no need to search the left branch, return to node x : 4.4.
18. Since both branches of the node x : 4.4 have been searched, return to node x : 3.4.
19. Since $dev_{total} = \sqrt{(4.8 - 3.4)^2 + (2.6 - 2.5)^2} \approx 1.40 > merit_{min} = 0.6$, no need to search in its left branch, return to node y : 2.5.
20. Since both branches of the root node y : 2.5 have been searched, the overall search is finished; the nearest neighbor is vector [4.8, 2].

In summary, different actions are taken according to different conditions:

Condition 1: If none of the current node's two branches have been searched, then search the branch that satisfies the partitioning criteria with the query event.

Condition 2: If one of the current node's two branches has already been searched, then determine whether the other branch satisfies $dev_{total} < merit_{min}$. If so, search in the other branch; if not, return to its parent node.

Condition 3: If both of the node's two branches have been searched, then return to its parent node.

Condition 4: If this node is a leaf, compute its distance to the query event. If this distance is smaller than the current $merit_{min}$ value, then update the $merit_{min}$ and record the leaf's position.

Note: The $merit_{min}$ is initialized with a very large number. The process of search is terminated if all branches of the root node have been searched or pruned.

2.2.3. Integrating maximum-likelihood estimation into k-d tree search—The example shown previously uses the Euclidean distance as the metric to judge the closeness of a query vector to a reference data sample. Other metrics that incorporate statistic properties of the data, such as likelihood, can also be used with k-d tree search. If the noise characteristics of the detector can be modeled accurately, the maximum likelihood method will yield more accurate results (Barrett *et al* 2009).

If the numbers of detected photons of a single gamma-ray interaction on the K light sensors are $n_0, n_1, \dots, n_{K-1}$, and if a reference sample's values in a leaf are $r_0, r_1, \dots, r_{K-1}$, then under assumption of Poisson statistics, the probability of the detected photons given the reference sample is:

$$P(n_0, n_1, \dots, n_{K-1} | r_0, r_1, \dots, r_{K-1}) = \prod_{i=0}^{K-1} \frac{e^{-r_i} r_i^{n_i}}{n_i!}. \quad (2)$$

After taking the logarithm of both sides:

$$\ln(P(n_0, n_1, \dots, n_{K-1} | r_0, r_1, \dots, r_{K-1})) = \sum_{i=0}^{K-1} -r_i + n_i \ln(r_i) - \ln(n_i!). \quad (3)$$

Dropping the constant term $\ln(n_i!)$, which does not change with r_i we define:

$$L(n_0, n_1, \dots, n_{K-1} | r_0, r_1, \dots, r_{K-1}) = \sum_{i=0}^{K-1} -r_i + n_i \ln(r_i). \quad (4)$$

Therefore, in order to find the nearest neighbor in terms of likelihood, we need to find the leaf with the set of $r_0, r_1, \dots, r_{K-1}$ that maximize equation (4). For each dimension (or light sensor) i , the corresponding L_i is:

$$L_i(n_i, r_i) = -r_i + n_i \ln(r_i)$$

and

$$L(n_0, n_1, \dots, n_{K-1} | r_0, r_1, \dots, r_{K-1}) = \sum_{i=0}^{K-1} L_i(n_i, r_i). \quad (5)$$

According to equation (5), at a node of the k-d tree, whose boundary value is b_i if we choose the branch that satisfies the partition criteria (if $n_i > b_i$, we search among r_i that satisfy $r_i > b_i$; if $n_i < b_i$, we search among r_i that satisfy $r_i < b_i$), it is easy to show that the highest possible L_i is achieved when $r_i = n_i$:

$$L_{i_{max}} = -n_i + n_i \ln(n_i). \quad (6)$$

If we choose the branch that violates the partition criteria, the highest L_i possible is achieved when $r_i = b_i$:

$$L_{i_{max}} = -b_i + n_i \ln(b_i). \quad (7)$$

In this case, if we already have a candidate result, whose L is $L_{current}$, the pruning can happen when a search has violated the partition criteria at a number of dimensions, such that their corresponding $L_{i_{max}}$ are limited by equation (7). For the remaining dimensions, even if their corresponding $L_{i_{max}}$ have achieved the highest values indicated by equation (6), the total L is still smaller than $L_{current}$ so there is no need to continue to search the branch (pruned).

This part shows the possibility of combining other metrics, such as maximum likelihood, into the k-d tree search method, which expands the flexibility of the algorithm.

2.3. Using k-d tree to estimate gamma-ray event position

Mean-detector-response functions (MDRFs) are the mean signal responses of the light sensors of a scintillator detector as functions of the gamma-ray interaction position, and they can be treated as maps of signal responses (of the light sensors). The position of an unknown event is found by finding among the many MDRF samples the champion that has the smallest distance with respect to the query event (nearest neighbor). The number of light sensors (M) is equal to the dimensionality of each event, and the number of candidate positions (N) in MDRFs corresponding to the number of entries of the k-d tree. Therefore, a k-d tree can be constructed with the MDRFs in a very natural way.

In order to account for the energy variations induced by scintillator's intrinsic energy resolution, such as Poisson statistics and electric noise, each event could be normalized by the summed signal value, before searching for its nearest neighbor in the k-d tree. If doing so, the MDRFs samples used to construct the k-d tree also need to be normalized.

The k-d tree was constructed by continuously subdividing the MDRFs' samples in k-dimensional space. The dimension to subdivide the MDRF samples was chosen randomly in this manuscript, and the boundary of the subdivision was chosen as the medium of the set of MDRF-sample values corresponding to that dimension.

2.4. Monte-Carlo simulation

To illustrate the utility of a k-d-tree search in the determination of gamma-ray interaction location, a simulation was carried out based on the edge-readout detector design (Li *et al* 2018). The size of the LYSO scintillator was chosen to be 50.8 mm × 50.8 mm × 3 mm, 20 SiPMs were modeled, each SiPM's size is 10.16 mm × 3 mm with an active area of 10 mm × 3 mm. The photon detection efficiency (PDE) was set conservatively to be 20% and the dark count rate was modeled as 1.0 MHz mm⁻². In addition, an excess noise factor of 1.21 was included. The refractive index of the optical gel and the SiPM window material were set to 1.5. Optical barriers were included in the crystal in accordance with the designs explored in Li *et al* (2018).

A perfectly thin gamma-ray beam of 511 keV photons were modeled to calibrate the detector at 252 × 252 positions, with step sizes in *x* and *y* directions of 0.25 mm. 1000 photoelectric interactions were modeled at each scanned position, then the MDRFs are determined by calculating the mean signal responses of each light sensor at each scanned location.

Projections created by illumination of 511 keV photons through a collimating mask with 7 parallel slits were also simulated; the spacing between the slits were 7.5 mm and the width of each resulting gamma-ray fan beam is 0.44 mm FWHM. There are 9.3×10^5 photoelectric events in total. The simulation and searching programs were carried out using C++.

2.4.1. Contracting-grid search method—Contracting-grid search (introduced in Hesterman *et al* (2010)) is then performed to estimate the interaction positions of the same 9.3×10^5 photoelectric interactions. The contracting-grid method utilized a 4 × 4 grid with initial length of 16 mm and a contracting ratio of 0.75.

2.4.2. k-d tree search method—The k-d tree was constructed with the MDRFs achieved from the calibration simulation procedure described previously. In this case, each entry in the k-d tree has 20 dimensions, and there are a total of 252 × 252 MDRF samples (corresponding to 63 504 leaves of the tree). And the interaction positions of events are estimated using the k-d tree search method introduced in section 2.2.

2.5. Experiment validation

The prototype detector presented in Li *et al* (2018) was also used for experimental measurements; the scintillation crystal was a slab of CsI(Tl) of size 27.4 mm × 27.4 mm × 3 mm. 16 Hamamatsu S13360–6050PE SiPMs (active area is 6 mm × 6 mm) were attached to the scintillator crystal to read from edges (4 SiPM sensors on each edge). Sixteen mechanically-drilled holes were introduced as optical barriers. The detector was calibrated

using a well collimated gamma-ray beam of 662 keV photons; the size of the calibration beam was $0.44 \text{ mm} \times 0.44 \text{ mm}$ FWHM, which was used to scan 81×81 locations with a step size of 0.25 mm, the corresponding MDRFs were calculated using the calibration data.

To test the positioning algorithm, slit beams of width 0.625 mm FWHM at 6 different locations were projected onto the detector; 50 000 gamma-ray interactions of 662 keV photons were recorded at each slit location.

2.5.1. Contracting-grid search—The contracting-grid method utilized a 4×4 grid with initial length of 7 mm and a contracting ratio of 0.75, a total of 3×10^5 events from six slit locations are searched.

2.5.2. k-d tree search method—The k-d tree was constructed with the MDRFs achieved from the experiment calibration dataset. In this case, each entry in the k-d tree has 16 dimensions (since there are 16 SiPMs), there are a total of 81×81 MDRF samples (corresponding to 6561 leaves of the tree). And the same set of events (as CG search above) are estimated using the k-d tree search method introduced in section 2.2.

3. Results

3.1. Monte-Carlo simulation

The simulated mean-detector response functions (MDRFs) are shown in figure 6.

The estimated projection image of these seven fan beams on the detector is shown in figure 7, estimated using the exhaustive search method. This result serves as the ground truth. It required 4292 s to finish the entire search, with an Intel i7-4770k CPU, single thread.

3.1.1. Contracting-grid search—Using the contracting-grid (CG) search method, the total search time is greatly reduced to 26 s, the resulting estimated projection image is shown in figure 8.

By comparing this image with figure 7, it is obvious that the contracting grid (CG) search method cannot provide enough precision with these complicated MDRF functions. Positioning errors are obvious in the image as the beam in the center is discontinuous, and some events are misplaced in some wrong places. Compared with the ground truth, the standard deviation is 1.9 mm. The reason that the contracting grid search method does not work well is due to the introduction of optical barriers, which create the MDRFs with much more complicated shapes, and there are many local-optimal positions. The search grid can be trapped into local-optimal positions and will thus occasionally fail to reach the global-optimal position. Three random events' likelihood maps are shown in figure 9.

3.1.2. k-d tree search—Using the k-d tree search method, the total search time is further reduced to 14 s. By comparing figures 7 and 10, their projection images are visually the same, and compared with the ground truth images of figure 7, the standard deviation of the slit images is 0.0 mm. This is not unexpected since the k-d tree search is actually another kind of exhaustive search, but with branch pruning to increase the searching speed.

3.2. Experiment validation

The experimental MDRFs achieved using the prototype edge readout detector (described in section 2.5) are shown in figure 11.

Figure 12 shows the estimated projection images of the six slit beams using the exhaustive search method, which are considered as the ground truth and it took 162 s to finish searching the 3×10^5 events.

3.2.1. Contracting-grid search—The six slit images achieved using the contracting-grid search method are shown in figure 13. The total search time is 4.5 s to estimate 3×10^5 events. If using the result of exhaustive search as ground truth, the CG search has a standard deviation of 1.3 mm (positioning error) due to occasional large errors in estimating some events' positions.

3.2.2. k-d tree search—The six images by position estimating using the k-d tree search method are shown in figure 14. The total search time is 1.8 s to estimate the same 3×10^5 events. If using the result of exhaustive search as ground truth, the k-d tree search has a standard deviation of 0.0 mm, confirming that the exactly same result is achieved compared with the exhaustive search method.

A comparison between the exhaustive search, contracting-grid search and k-d tree search in terms of accuracy and efficiency for the examples in the paper is summarized in table 1.

4. Discussions and conclusions

Both the simulation and experimental results show that the k-d tree search method can provide exactly the same precision as the exhaustive search with greatly-improved search speed. Compared with the contracting-grid search method, the k-d tree search method is more robust when searching in MDRFs with complicated merit function map (such as likelihood map) and also the search time is further reduced.

Since the k-d tree search method actually has exactly the same accuracy as exhaustive search method, the nearest neighbor of a query event is guaranteed to be found in decently short time (average search time is $O(\log_2(N))$), where N is the number of entries in the reference dataset. The algorithm is also relatively simple to implement, and is robust to adapt to different situations (such as searching a detector with complicated MDRFs or using a different merit function to search). Moreover, this concept can be further extended into applications in other fields, such as locating interaction events in high-energy physics experiments (alpha and beta particles, energetic nucleus fragment or even neutrons), whenever a critical requirement on achieving both 'accuracy and speed at the same time' is needed.

Still, similar to many other reference-data-based search methods, the apparent weakness of the k-d tree search is the need of acquiring calibration data to construct the k-d tree, which could be very time-consuming. Also, the construction of the k-d tree requires a little bit more memory to store the nodes of the k-d tree (since the nodes requires less memory than

leaves, the increase of memory use is small), while the leaves store the whole reference dataset. In addition, due to the many branches in the code of k-d tree search (many 'if' and 'else' statements, which create branches in the code), applying it on GPU is more challenging. In addition, the possible need to transport raw data (all readings of the light sensors of each event instead of its interaction position and energy) from detector modules to local computers will pose more harsh technical challenges on the communication bandwidth and data storage capacity. Despite these challenges, the high accuracy and fast searching capability of k-d tree search method facilitate its broad applications in SPECT/PET and high energy physics detectors, where fast and reliable localization of interaction events is required.

Acknowledgment

This work was partially supported by NIH/NIBIB grants P41-EB002035 'The Center for Gamma-Ray Imaging' and 5R01EB000803.

References

- Aarsvold J, Mintzer R, Matthews K, Yasillo N, Ordóñez C and Chen CT 1995 Implementations of maximum-likelihood position estimation in a four-PMT scintillation detector IEEE Nuclear Science Symp. and Medical Imaging Conf. Record vol 3 (IEEE) p 1811
- Anger HO 1958 Scintillation camera Rev. Sci. Instrum 29 27–33
- Barrett HH, Hunter WC, Miller BW, Moore SK, Chen Y and Furenlid LR 2009 Maximum-likelihood methods for processing signals from gamma-ray detectors IEEE Trans. Nucl. Sci 56 725–35 [PubMed: 20107527]
- Bentley JL 1975 Multidimensional binary search trees used for associative searching Commun. ACM 18 509–17
- Hesterman JY, Caucci L, Kupinski MA, Barrett HH and Furenlid LR 2010 Maximum-likelihood estimation with a contracting-grid search algorithm IEEE Trans. Nucl. Sci 57 1077–84 [PubMed: 20824155]
- Lewellen TK. 2008; Recent developments in PET detector technology. Phys. Med. Biol. 53:R287. [PubMed: 18695301]
- Li X, Ruiz-Gonzalez M and Furenlid LR 2018 An edge-readout, multi-layer detector for positron emission tomography Med. Phys 45 2425–38 [PubMed: 29635734]
- van Dam HT, Seifert S, Vinke R, Dendooven P, Lohner H, Beekman FJ and Schaart DR 2011 Improved nearest neighbor methods for gamma photon interaction position determination in monolithic scintillator PET detectors IEEE Trans. Nucl. Sci 58 2139–47
- Yonggang W, Junwei D, Zhonghui Z, Yang Y, Lijun Z and Bruyndonckx P 2011 FPGA based electronics for PET detector modules with neural network position estimators IEEE Trans. Nucl. Sci 58 34–42

Figure 1.
Example of a 1D tree, sorting eight numbers in ascending order. The ellipses are nodes and boxes are leaves.

Figure 2.

The 2D tree organizing the 16 2D vectors. Each leaf in the tree represents a 2D vector.

Figure 3.
The 2D map representing the 16 2D vectors. Each dot is one vector.

Figure 4.
The 2D map (figure 3) segmented by the 2D tree. Each line represents a node in the 2D tree.

Figure 5.
A 3D tree segmenting a 3D space into multiple subspaces.

Figure 6.
The MDRFs achieved by simulation of an ideally thin gamma-ray beam of 511 keV gamma-ray photons scanning the detector at 252×252 locations.

Figure 7.

The estimated projection image of the seven fan beams created by slits of width 0.44 mm, using the exhaustive search method. Variance is large near detector edges partially due to undesirable total internal reflection (TIR) on detector edges (Liet *al* 2018).

Figure 8.
The estimated projection image of seven fan beams, using the contracting-grid search method.

Figure 9.
The likelihood map of three random events. To search for a global-optimal position is extremely challenging due to the appearance of many local-optimal positions.

Figure 10.
The estimated projection image of the seven fan beams using k-d tree search method.

Figure 11.
The measured MDRFs of the prototype detector.

Figure 12.
The six slit projection images estimated using the exhaustive search method.

Figure 13.
The six slit projection images estimated using the contracting-grid search method.

Figure 14.
The six slit projection images estimated using the k-d tree search method.

Table 1.

Accuracy and efficiency comparison between the three search methods.

		Exhaustive search	CG search	k-d tree search
Simulation	Standard deviation from ground truth	N/A	1.9 mm	0.0 mm
	Search time for 9.3×10^5 events	4292 s	26 s	14 s
Experiment	Standard deviation from ground truth	N/A	1.3 mm	0.0 mm
	Search time for 3.0×10^5 events	162 s	4.5 s	1.8 s