Arkade: k-Nearest Neighbor Search With Non-Euclidean Distances using GPU Ray Tracing

Durga Mandarapu
Purdue University
West Lafayette, IN, USA
dmandara@purdue.edu

Vani Nagarajan
Purdue University
West Lafayette, IN, USA
nagara16@purdue.edu

Artem Pelenitsyn
Purdue University
West Lafayette, IN, USA
apelenit@purdue.edu

Milind Kulkarni
Purdue University
West Lafayette, IN, USA
milind@purdue.edu

ABSTRACT

High-performance implementations of k-Nearest Neighbor Search (kNN) in low dimensions use tree-based data structures. Tree algorithms are hard to parallelize on GPUs due to their irregularity. However, newer Nvidia GPUs offer hardware support for tree operations through ray-tracing cores. Recent works have proposed using RT cores to implement kNN search, but they all have a hardware-imposed constraint on the distance metric used in the search—the Euclidean distance. We propose and implement two reductions to support kNN for a broad range of distances other than the Euclidean distance: Arkade Filter-Refine and Arkade Monotone Transformation, each of which allows non-Euclidean distance-based nearest neighbor queries to be performed in terms of the Euclidean distance. With our reductions, we observe that kNN search time speedups range between 1.6x-200x and 1.3x-33.1x over various state-of-the-art GPU shader core and RT core baselines, respectively. In evaluation, we provide several insights on RT architectures' ability to efficiently build and traverse the tree by analyzing the kNN search time trends.

CCS CONCEPTS

• **Computing methodologies** → **Ray tracing**; *Graphics processors*; • **Information systems** → **Nearest-neighbor search**; • **Theory of computation** → *Nearest neighbor algorithms.*

KEYWORDS

GPU Ray Tracing, k-Nearest Neighbor Search, Non-Euclidean Distances

https://doi.org/10.1145/3650200.3656601

1 INTRODUCTION

k-Nearest Neighbor Search (kNN) is the problem of finding points similar to a query point based on a desired distance function. Several commonly used distance functions include Euclidean distance (L^2 norm), Manhattan distance (L^1 norm), Chebyshev distance (L^∞ norm), Minkowski distance (L^p norm), and Cosine (or Angular) distance. kNN is used in diverse applications, including point cloud registration [42], facial recognition [26, 30], recommendation systems [1], and more.

Due to the computational intensity and the wide applicability of kNN, many optimization techniques have been proposed in this space: *tree-based approaches*, such as kd-tree or ball tree [14, 31, 51]; graphs, such as proximity graphs or kNN graphs [11, 28, 53]; hashing, such as locality sensitive hashing [21, 22, 41]; quantization, such as product quantization codes [3, 17, 25].

Tree-based approaches to kNN work better and provide logarithmic guarantees in lower dimensions [5, 10]. Low-dimensional data (two to three dimensions) is predominant in several applications, such as spatial query processing [39] and astronomical data [43], where tree-based approaches have gained popularity. However, tree-based approaches can not be efficiently accelerated using GPUs, unlike the non-tree indexing methods. Tree-based implementations on GPU run kNN queries in parallel by mapping each query to a GPU thread that traverses the tree. These traversals are highly *irregular:* different traversals touch different parts of the tree, leading to control divergence, and the tree itself can be scattered around memory, leading to memory divergence [15]. Nevertheless, several recently proposed algorithmic approaches improve GPU efficiency for tree traversals, leading to fast nearest neighbor searches on low-dimensional data [15, 19, 34, 51].

Modern GPUs do not just contain the shader cores used by prior approaches, but also ray-tracing (RT) cores. RT cores are built to accelerate ray tracing [2, 23, 37]: identifying which objects in a scene are intersected by rays cast from a source such as the viewer's eye. Ray-tracing is an inherently irregular problem, and these ray tracing cores perform *hardware accelerated tree traversals*: they build a spatial tree called a *bounding volume hierarchy* over the objects in a scene, then each ray traverses that tree to find the objects it intersects. While ray tracing is a highly specific algorithm, and it may seem that RT cores cannot be used to solve other problems, prior work has shown that by carefully constructing the objects in a scene and properly defining the rays, it is possible to find solutions to non-ray tracing problems by *reducing* them to ray tracing [9, 33, 48, 54]. In particular, several prior papers have shown how to reduce kNN to ray tracing [9, 33, 52, 54] (see Subsection 7.1,2.4).

Unfortunately, the existing kNN approaches on RT cores are all based around a single reduction that inherently uses the Euclidean distance (L^2 norm) as the desired distance function. This limitation is unsurprising, as the RT cores arrange objects in a scene according to the Euclidean distance. However, one cannot merely use L^2 nearest neighbor as a proxy for other distance functions. For example, an object being at a particular Euclidean distance from the point of interest says nothing about a non-L^2 distance function such as the

Angular distance between them (Figure 1). In practical applications like street maps and astronomical settings where Euclidean distance falls short in conveying essential information, non-L^2 distances are needed (see Section 2.1 for more detail). However, prior work on RT cores cannot address these non-L^2 distance requirements.

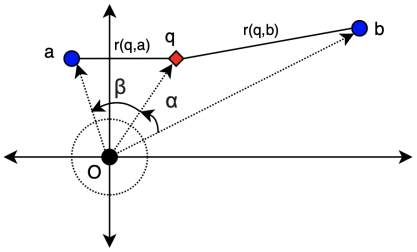


Figure 1: Euclidean and Angular distances: a and b are data points, q is a query point, and O is the point of reference. $L^2(q,a)\ \ddot{Y}\ L^2(q,b) \implies \beta\ \ddot{Y}\ \alpha$

To support non-Euclidean kNN queries, we make a key observation that the prior kNN reductions to RT cores do not solve the nearest neighbor problem *directly* [9, 33, 54]. Instead, the reductions accelerate an *r-bounded distance query*: find all points within a distance r of a query point q, according to their Euclidean distance. Similarly, instead of solving kNN problem for other distance functions on RT cores directly, in this paper, we show how to reduce kNN searches in other distances to the r-bounded Euclidean distance search and implement the reductions for RT cores.

Contributions

This paper introduces *Arkade*, a suite of two general reductions: *Filter-Refine* (RT) and *Monotone Transformation* (MT), each allowing non-Euclidean distance-based nearest neighbor queries to be performed on RT cores. In particular, we contribute the following.

(1) *Arkade FR reduction* performs a generic distance-based kNN search using RT cores by decoupling the kNN search into *Filter* and *Refine* phases and adapting a tree-based kNN algorithm for distances besides the Euclidean distance (Section 3). The reduction utilizes geometric properties common in some popular distance functions, such as L^p distances.
(2) *Arkade MT reduction* enables RT-based acceleration of kNN search for distance functions that do not hold the geometric properties favored by Arkade FR reduction (Section 4). The reduction transforms the input such that the original order of distances between the data points is preserved. Important examples of such distances are cosine distance or angular distance.
(3) Evaluation of Arkade (FR and MT) implemented as stand-alone applications using RT cores of the NVIDIA GeForce RTX 4060 Ti GPU (Section 6). Our reductions show speedups of 1.6x-200x and 1.3x-33.1x over various state-of-the-art GPU shader core and RT core baselines, respectively.

Arkade (aRKaDe) derives its name from the three parameters this paper considers — radius (r), number of neighbors (k), and distance function (D).

2 BACKGROUND

2.1 k-Nearest Neighbour Search

We define the kNN search problem in Definition 1 since there are several variants of kNN. Importantly, the particular distance function D is a parameter.

Definition 1 (k-Nearest Neighbor Search). *Given a query point $q \in \mathsf{R}^d$, a set of data points, $A \subseteq \mathsf{R}^d$, a value $k \in \mathsf{N}$, and a distance function $D : \mathsf{R}^d \times \mathsf{R}^d \to \mathsf{R}$, the generalized k-nearest neighbor problem finds a result set of points, $T \subseteq A$, that contains the closest k points to q according to D.*

The naive way of performing kNN is computing the distance between q and all of the points in A and ordering them by D, an $O(n \log k)$ process[1] ($|A| = n$). Tree-based approaches [14, 31, 51] can avoid comparing q to every point in A by efficiently indexing the points in A using a tree and pruning the search space, resulting in an $O(\log n \log k)$ algorithm. However, the trees built by RT cores use L^2-based pruning [9, 33, 48, 54], and hence this approach *only* works if the distance function D is the L^2 distance.

Other distance functions. The key focus of this paper is using RT cores, which are inherently tied to L^2 distances, to solve non-L^2 distance problems. This subsection summarizes some of these non-L^2 distance functions.

In 2-dimensional space, we recall the set of functions that give the distance between point a and b based in L^p spaces [6] as follows, where a_x, a_y are x, y coordinates of point a, and $|.|$ represents the absolute value:

$$L^p(a,b) = (|a_x - b_x|^p + |a_y - b_y|^p)^{\frac{1}{p}}, p \in \mathsf{R} \geq 1$$
$$L^\infty(a,b) = max(|a_x - b_x|, |a_y - b_y|)$$

While L^p norms use Cartesian coordinates, distances such as angular distance, cosine distance, inner product, or dot product use spherical coordinates [46]. Angular distance is the shorter angle between two vectors, while cosine distance or cosine similarity is the cosine function applied to this angle. The inner product or dot product is the same for vectors, and these are, in turn, the same as cosine distance when the vectors are of unit length. Because cosine distance measures how similar two vectors are, it is highly useful in recommendation systems.

This paper confines its scope to 2- and 3-dimensional spaces because RT cores operate solely within these dimensions. The utility of non-Euclidean distances in these lower dimensions remains evident in several domains, such as geospatial applications and astronomy. For instance, consider street maps, where the determination of nearest points of interest hinges on the ordering of their *Manhattan distance* from the query point location since the data points in a city usually adhere to taxicab geometry. Similarly, visually nearby stars are identified with cosine distance in 3 dimensions instead of Euclidean distance: the three stars in Orion's belt are not L^2-close together—they are approximately 2000, 1200, and 700 light years away from Earth—despite being visually adjacent.

[1]The $\log k$ term comes from efficiently maintaining distances of the top k neighbors

2.2 Ray Tracing Architecture

Ray tracing is a graphics rendering algorithm where rays are modeled from a starting point as a source and followed (traced) till they hit the objects in a scene. The fundamental operation in ray tracing is computing *ray-object intersections*: for a given ray, what object(s) does the ray intersect? This problem shares some features with nearest-neighbor search: the naïve algorithm compares a ray to each object in a scene but can be accelerated using a spatial tree to prune the space. In the case of ray tracing, this spatial tree is called a *bounding volume hierarchy* (BVH) [45], and the RT architecture on modern GPUs provides acceleration for building and traversing this spatial tree.

The RT architecture employs both RT cores and shader cores (also called streaming multiprocessors) to accelerate various stages of the ray-tracing pipeline. Optimized drivers build a BVH bottom-up by enclosing each object in an *axis-aligned bounding box* (AABB) and grouping AABBs such that several AABBs can be enclosed in a larger AABB. Eventually, the overall scene is enclosed in a single AABB. RT cores recursively traverse the BVH tree to compute ray-object intersections. In particular, if a ray intersects an AABB, then the enclosed bounding boxes will be tested next. The process continues until it reaches leaf AABBs. At that point, the shader cores execute user-defined code to determine whether the ray intersects the object contained in the AABB. If an intersection is found, another user-specified code is called.

2.3 Programming and Execution Model

Optix [35] is a programming interface that provides access to the entire RT architecture. This interface allows the user to write traditional shader programs that are executed on the shader cores and leverage the RT hardware for BVH construction, traversal, and, if applicable, intersection testing. Optix allows the user to specify user-defined geometries, which we use to represent neighborhoods in non-L^2 distances. Important Optix kernels that we use are RayGen and Intersection. RayGen kernel creates rays with user-specified parameters such as the origin, direction, and length of the ray. It then calls for BVH traversal and intersection testing. For user-defined geometries, the user is required to provide a custom intersection test for ray-object intersections in the form of an Intersection kernel.

Geometric Objects. For a distance function D, all the points that are at a D-distance of r could be described by a geometric object. For example, if the distance function is L^1 norm, then the geometric object is a square rhombus in 2D space and a square bi-pyramid in 3D space. Similarly, if the distance is L^2 norm, then the geometric object is a circle in 2D space and a sphere in 3D space. A geometric object simply refers to a geometry whose periphery contains points that are equidistant from the center of the geometry. The geometric objects are then placed inside AABBs. With the Optix interface, it is up to the user to define the distance function of geometric objects, so these geometric objects are also called user-defined or custom-defined geometries.

Limitations. There are several limitations when re-purposing RT architecture to perform a non-RT task. First, we are limited to using data with three dimensions. Second, the BVH built by the RT architecture is not accessible nor programmable in any kind by a user. There is no available information on how the BVH is constructed or traversed. Third, during the traversal, the Optix interface notifies the user only when a successful ray-AABB intersection occurs. We do not know the actual number of AABBs that are tested during the traversal or the actual traversal path. Fourth, even after successful mapping, it is hard to assess the resource utilization of our mapping and identify opportunities to optimize the hardware usage due to inadequate support from the profilers.

2.4 RT-kNN: kNN on RT architecture

Accelerating a non-RT problem with RT cores requires defining several components that we call a *reduction*. A reduction defines a scene with objects and rays such that the hardware-accelerated ray-AABB intersection detection encodes a partial or complete solution to the initial non-RT problem. The reduction should define how to decode that solution.

In particular, the reduction of kNN to RT only aims to accelerate a part of the problem, which is the *r-bounded distance query*. We refer to this reduction as 'RT-kNN' for the rest of the paper. Figure 2 shows how the RT-kNN reduction (on the right) solves a flipped-around version of the conventional kNN algorithm (on the left). It tries to find if the query point is at a distance less than or equal to r to a data point rather than finding the data points that are within a distance of less than or equal to r to a query point. To find the neighbors of query points, RT-kNN reduction models the data points as spheres, the query points as rays, and the neighbor identification as an intersection of the corresponding ray with the spheres, as explained in more detail below [52].

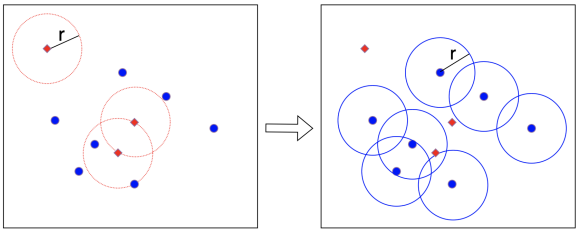


Figure 2: RT-kNN reduction (right) finds all query points within radius r to data point unlike the conventional kNN algorithm (left) that finds all data points within radius r to the query point. Blue circles and red rhombus represent data and query points, respectively.

Given a set of data points A and a set of query points Q, the RT-kNN reduction builds spheres of radius r centered around all data points in A, as shown in the right part of Figure 2. To find the neighbors, the reduction launches point rays from every query point. A point ray is a ray whose length is a very small positive number. If a point ray cast from the point $q \in Q$ as the source intersects with the sphere of radius r and center $a \in A$, then it means that the point q is present inside the sphere and so the Euclidean distance between points a and q is at most r.

3 FILTER-REFINE

In this section, we show how to map k-nearest neighbor search for distance functions beside L^2 norm to a ray tracing problem. For

this purpose, we use a general framework called Filter-Refine. In particular, we formulate *Arkade Filter-Refine reduction* (Subsec. 3.1) and prove its correctness (Subsec. 3.2).

Filter-Refine is a two-step selection framework for search problems [50]. First, we *filter* a subset of the possible candidates from the data points and then *refine* this subset to produce a result that answers the original search query, exactly or approximately. Drawing on the principles of this framework, we devise a reduction that breaks down kNN search for a generic distance function into Filter and Refine phases and maps these phases to operations performed by the RT architecture.

3.1 Arkade Filter-Refine Reduction

Assume an arbitrary distance function D. To find the nearest k points within the D-distance of r to a query point q, the Arkade Filter-Refine (FR) reduction employs the following *Filter* and *Refine* phases.

(1) *Filter Phase* finds all the candidate data points that are within the D-distance of r, by mapping the data points and query points to a ray tracing scene.
(2) *Refine Phase*, once the candidates are filtered, sorts them according to their D-distance to the query point q and finds the k nearest neighbors.

The first step of the reduction involves solving the r-bound query problem, which is where the RT architecture comes in. The hardware accelerates the search process of candidates since we encode them as a part of the ray tracing problem. In particular, to find all the data points within a D distance of r from the query points, the reduction builds specific distance function geometric objects centered at data points and launch point rays originating from query points. The ray traverses the BVH to find the candidates that will be passed to the Refine phase.

In Figure 3, part (a) on the left shows data points and query points colored in blue and red, respectively. When an RT core finds an intersection, the intersection is with the AABB that contains the geometric object, rather than the geometric object itself. To ensure that there are no false positive candidates, the ray-AABB intersections are further filtered to remove the data points where the query point lies inside the AABB but outside the geometric object. Part (b) of Figure 3 shows how the Filter phase first uses RT cores to get the AABBs that a point ray intersects and then uses the shader cores to perform the intersection with the geometric object present inside these intersected AABBs. AABBs and geometric objects are represented by squares and circles, respectively. The green AABBs or circles are the ones selected, while the blue ones are not.

The second step of the reduction, the Refine phase, processes the candidates that are passed on from the Filter step. In particular, the candidates are ordered to select the nearest k data points to the query point. Part (c) in Figure 3 shows that the blue and green points are the candidates processed in the refine phase, out of which only the green points are selected as the top-k neighbors.

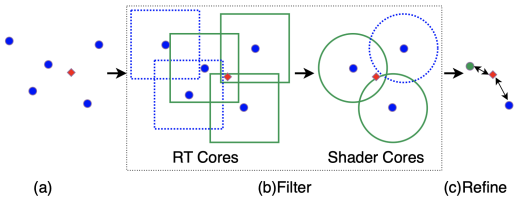


Figure 3: Filter-Refine: (a) map points to RT scene, (b) RT cores filter AABBs and shader cores filter geometric objects, (c) refine candidates to select $k = 1$ nearest neighbors.

We present *Arkade Filter-Refine reduction* in algorithm 1. In Line 1, AABBs corresponding to each data point are defined. If D is L^1 norm, then the geometric object is a square rhombus and the AABB with a side of length $2r$ should be defined to tightly fit the rhombus. It is up to the user to decide how big of a bounding box is needed to render the desired geometry. However, the tighter, the better. In line 2, an Optix API call is made to build the BVH on the defined AABBs. The constructed BVH is not returned to the user but is available for the RT architecture to traverse. In line 3, an Optix API call is made to launch point rays from each query point. From line 4, the neighbor search starts. In lines 4-5, RT cores perform the BVH tree traversal of the ray and return the AABB intersection when a ray is found to intersect with AABB. Lines 5-9 and lines 10-12 indicate the Filter and Refine phases respectively. Lines 6-7 extract the data point, which is the center of geometry inside the hit AABB, and the query point, which is the source of the point ray hitting the AABB. In line 8, we compute the D-distance between them. In line 9, we filter out all the data points that are farther than a D-distance of r. Lines 10-12 refine the selected candidates and store the top k closest points.

Algorithm 1 Arkade Filter-Refine Reduction

Input: Training set A, Query set Q, distance function D, r, k
Output: $\forall q \in Q$, top k neighbors of q within D distance r

1: $\forall a \in A$, define AABB on the geometry centered at a
2: construct BVH on all the AABBs
3: $\forall q \in Q$, launch point ray at q
4: **while** each ray is traversing BVH **do**
5: **if** RT cores return ray-AABB intersection **then**
6: $a \leftarrow geometry.center$ • data point
7: $q \leftarrow ray.origin$ • query point
8: $w \leftarrow D(a, q)$
9: **if** $w \leq r$ **then**
10: **if** w Ÿ $max(neighbors(q).distance)$
11: or $|neighbors|$ Ÿ k **then**
12: $neighbors.insert(a, w)$

In the Algorithm 1, the Filter and Refine phases are interleaved. Instead of storing all the candidates from the Filter phase, each candidate is refined on the go. Once the RT core finds a point that is within r distance, the reduction uses the shader cores to dynamically update the list of k nearest neighbors, and return the control to RT cores to resume the search for candidates.

The Arkade FR reduction presented above is a generalization of the RT-kNN reduction and uses RT cores in a novel way. RT-kNN ships spheres to RT cores because an r, L^2-ball (Def 2) is exactly a sphere. Similarly, for a distance D, we need to build geometric

objects customized to the distance function to represent an r, D-ball. Our key observation with Arkade Filter-Refine is that the RT architecture can process custom geometric objects.

Although Arkade FR reduction depends on a more advanced feature of RT cores, it finds a way to stay agnostic to the inherent property of RT cores, which only understand L^2 distances. The distance fixed by the hardware does not impact the core idea of the reduction—using point rays to find the kNN candidates. A point ray intersects with an object containing it as long as the query point is present inside the geometric object centered at a data point, and this does not depend on the hardware-defined metric.

Arkade FR reduction is generic over the distance function D. Hence, the effectiveness of this reduction depends on the distance function and consequently, the geometric objects that will be built centered at the data points. If a distance function geometric object is such that the Filter phase forwards most of the data points as kNN candidates, Arkade FR reduction is not useful. Because it has to process the unnecessarily large number of candidates in the Refine phase and this might not be better than a linear scan. An example of such a distance function is cosine distance. We address how to perform cosine distance-based kNN search in Section 4.

3.2 Correctness of Arkade FR Reduction

To prove the correctness of Arkade FR reduction, we first introduce r, D-ball in Definition 2 and then formally define Filter and Refine phases in Definitions 3 and 4 respectively.

Definition 2 (r, D-ball centered at a point b, $B_D(b, r)$). *r, D-ball in R^d centered at a point b is a set of points a that are within a D-distance of r from b:*

$$B_D(b, r) = \{a \mid a \in \mathsf{R}^d, D(b, a) \leq r\} \tag{1}$$

Definition 3 (Filter). *Given a training set of data points A, a set of query points Q, and a positive real number r, the Filter phase outputs all the data points in A that are within a D-distance r of each query point $q \in Q$ (i.e. $A \cap B_D(q, r)$).*

Definition 4 (Refine). *Given a natural number k and a set of points in $B_D(q, r)$ for each query point $q \in Q$, the Refine phase outputs the k closest points to q according to the D distance.*

Theorem 1 (Correctness of Arkade FR reduction). *Given a training set of data points A, a set of query points $Q, q \in Q$, a natural number k, a positive real number r, and a distance function D, Algorithm 1 computes the k nearest data points of q within a D-distance of r from q.*

Proof. We first show that any point removed by the **Filter** phase of Algorithm 1 is not inside $B_D(q, r)$. Then, we show that any point not within the k closest points to q gets removed by the **Refine** phase. We use these two claims to conclude that the set of points returned by Algorithm 1 is exactly the k nearest neighbors to q within $B_D(q, r)$.

We first claim that the **Filter** phase does not remove any points inside $B_D(q, r)$. Let a be a point in A and G_a be the AABB centered at a. Notice that by construction, the r, D-ball centered at point a, $B_D(a, r)$ is contained in the AABB G_a (i.e., $B_D(a, r) \subseteq G_a$). The point a is removed by the **Filter** phase exactly when the point ray originating from q does not intersect G_a, which by the discussion in Section 2.4 means q is not a point on or inside G_a, so q is not an element of $B_D(a, r)$ which implies that the D distance between q and a is greater than r.

$$q \not\in G_a \implies q \not\in B_D(a, r) \implies D(a, q) > r$$

However, this also implies that $B_D(q, r)$ does not contain a (i.e., $a \not\in B_D(q, r)$) and hence a should be removed.

Now, we claim that any point not within the k nearest neighbors of q gets correctly removed by the **Refine** phase. Let $a \in A$ be a point not removed by the **Filter** phase (so $D(a, q) \leq r$) but such that a is not one of the k nearest neighbors to q. This means that there must be k *other* points $a_1, a_2, \ldots, a_k$ such that the farthest of k neighbors is closer to q than a is (i.e., $a_i < a$ and $\max_i D(a_i, q) \leq D(a, q)$). Then a gets removed on line 12 of Algorithm 1.

Since Algorithm 1 does not remove any points that should be kept, and does not keep any points that should be removed, its output is exactly the k points in $B_D(q, r)$ that are closest to q. □

4 MONOTONE TRANSFORMATION

This section introduces a new reduction, Arkade Monotone Transformation (MT), that handles some metrics outside L^p better than the Arkade FR reduction. The L^p distance functions, the primary focus of Section 3, share an important property: their r, D-balls correspond to geometric shapes that can be efficiently represented and processed by RT cores. But this property fails for some important distances, e.g. the cosine distance. To accommodate some of such distances (including cosine), the Arkade MT reduction uses monotone transformations to reduce kNN in the given metric to kNN in L^2. The resulting kNN problem is solved with the well-established L^2-distance based RT-accelerated search using spheres [9], which is implemented as the L^2-instance of Arkade FT.

Arkade MT reduction is based on the following property.

Definition 5 (Monotonicity of distance functions). *A distance function D on R^n is* monotonically increasing *(resp.* decreasing*) at a point $q \in \mathsf{R}^n$ if there exists a transformation $f\colon \mathsf{R}^n \to \mathsf{R}^n$ such that for any two points a_1 and a_2 in R^n, if q is closer to a_1 than a_2 in terms of the distance D, then after applying the transformation, $f(q)$ is still closer to (resp. further from) $f(a_1)$ than $f(a_2)$ in terms of L^2 distance:*

$$\begin{aligned} &D(q, a_1) < D(q, a_2) \implies \\ &\qquad L^2(f(q), f(a_1)) < L^2(f(q), f(a_2)) \\ &\quad (\text{resp. } L^2(f(q), f(a_1)) > L^2(f(q), f(a_2))). \end{aligned}$$

A distance function D is monotonically increasing (decreasing) *if it is monotonically increasing (resp. decreasing) at every point $q \in \mathsf{R}^n$.*

The Arkade MT reduction transforms the input points such that the ordering of the points according to the given distance function is preserved when the transformed points are ordered according to the L^2 distance. The preservation of the ordering can be either positive or negative i.e., the ordering of the transformed points is either the *same* or the *reverse* as that of the original points. Now, we formally define the Arkade MT reduction in Definition 6.

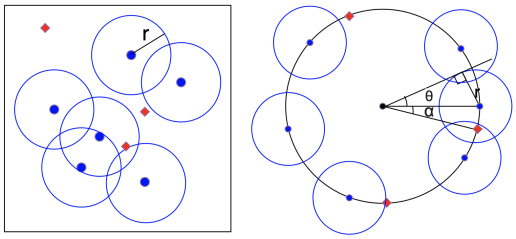


Figure 4: Arkade Monotone Transformation reduction: normalizing points for the cosine distance. Data and query points are marked with, blue and red colors, respectively. L^2-based Arkade FR reduction can only be applied after the Monotonic Transformation (normalization) to get the correct cosine distance-based kNN.

Definition 6 (Arkade Monotone Transformation Reduction). *Given a training set of data points A, a set of query points Q, $q \in Q$, a natural number k, a monotonic distance metric D and the corresponding transformation f, Arkade Monotone Transformation reduction applies f to points in A and Q and performs the Arkade Filter-Refine reduction with L^2 distance to find k nearest neighbors of every query point from the set of data points.*

Cosine Distance. As shown in Figure 1, the cosine distance between arbitrary vectors does not correlate with the Euclidean distance between vectors' endpoints. To introduce a correlation between the cosine and Euclidean distances, the transformation f we apply is normalization. In Equation 2, the normalization multiplier divides each of the coordinate components a_x, a_y, and a_z of the vector a by the vector's magnitude ω.

$$f: (a_x, a_y, a_z) \rightarrow \left(\frac{a_x}{\omega}, \frac{a_y}{\omega}, \frac{a_z}{\omega}\right), \omega = \sqrt{a_x^2 + a_y^2 + a_z^2} \qquad (2)$$

When the vectors are normalized, the end points (data points) fall on the unit circle. The cosine distance between two vectors is the same as the cosine distance between their normalized versions. Let α be the angle between the query point q and data point a. Because q and a are normalized, they have a unit magnitude. The relation between their Euclidean and cosine distances would be the following:

$$L^2(q,a) = \sqrt{||q||^2 + ||a||^2 - 2||q||||a||cos(\alpha)}$$
$$= \sqrt{1^2 + 1^2 - 2 \cdot 1^2 cos(\alpha)} = \sqrt{2 - 2cos(\alpha)}.$$

According to the above relation between the Euclidean and cosine distances in the normalized space, as the cosine distance between the vectors q and a increases, the angle (α) they make at the center decreases, and so the endpoints (data points) of the vectors move closer to the circle, which makes the Euclidean distance between the endpoints smaller. Therefore, the cosine distance decreases as the Euclidean distance between two data points increases. While cosine distance ordering is negatively preserved (reversed) by Euclidean distance ordering, the Angular distance (α) ordering, is positively preserved by Euclidean distance ordering.

In Figure 4, the left and right pictures represent the original and transformed points, respectively. The left part shows that simply building L^2 distance-based spheres and using the Arkade FR reduction with L^2 distance will not give us correct neighbors according to the cosine distance. The right part signifies that since the normalization preserves the ordering, the Arkade MT reduction can feed the normalized points to the Arkade RF reduction to get the correct k nearest neighbors according to the cosine distance.

5 DISCUSSION

5.1 Inclusion property to generalize RT-kNN

The closest prior work, RT-kNN (Section 2.4), is limited to the L^2 distance. In this subsection, we define the inclusion property of a distance function, which allows us to generalize RT-kNN to other metrics. We also explain why this generalization cannot perform better than Arkade and will typically perform worse.

The RT-kNN reduction cannot solve the problem with an arbitrary metric D without certain alterations. For example, consider the L^∞ distance and the r-bounded kNN problem. If we supply the RT-kNN reduction with the radius r, the candidates outside of the circle but inside the square (an L^∞ "circle") will not be found (subfigure 5(a)) and become false negatives. On the other hand, we could try to supply the RT-kNN reduction with a radius r' larger than r (e.g. $r' = \sqrt{2}r$, subfigure 5(b)). In that case, the k closest neighbors computed according to the L^2 distance are not the same as that of the L^∞ distance. In particular, point a is further than any point in the space between the square and circle according to the L^2 distance, but the reverse is true according to the L^∞ distance. Hence, RT-kNN reduction may have to exclude point a from the resulting set of nearest neighbors, while the point should be in the set according to the L^∞ metric. Point a becomes a false negative in this case. Note, that this issue can be avoided if we increase k to some k'. In general, by choosing r' and k' arbitrarily larger than the given r and k, we can use the RT-kNN reduction to perform the kNN search based on a non-L^2 distance D, although it may take extra time to test the candidates that could have been discarded early.

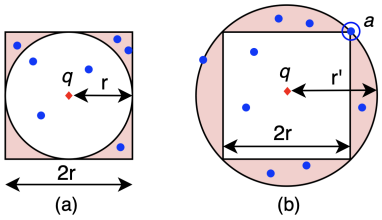


Figure 5: RT-kNN reduction can be extended to perform non-L^2-based kNN with a larger r and k using Inclusion property (Definition 7).

We call the property of a distance D that allows us to find a finite r' the *Inclusion Property* (Definition 7). This property states that it is possible to find the kNN candidates according to distance D as kNN candidates according to L^2 distance. For example, consider the L^∞ distance and $r = 1$. By the inclusion property, it is possible to construct a finite-sized sphere S of radius $r' = \sqrt{2}$ according to the L^2 norm such that all the points that are within a distance r according to L^∞ norm will fall inside sphere S.

Definition 7 (L^2-inclusion property of distance D). *A distance function D holds the L^2-*inclusion property *if for any point b and positive real number r there exists a positive real number r' such that the r, D-ball around point b is contained in the r', L^2-ball around point b.*

$$\exists r' : B_D(b, r) \subset B_{L^2}(b, r'). \tag{3}$$

However, the inclusion property is only helpful when the sphere of radius r' efficiently filters the candidates. In the case of distances where r' is too large that it includes all the points in the dataset, the search process is no better than a linear scan. Moreover, the inclusion property only addresses how to choose r' but not k' for given r and k, respectively. Currently, we choose k' through trial-and-error. We keep on incrementing k' until the RT-kNN can find all the k actual nearest neighbors.

5.1.1 RT-kNN reduction vs Arkade FR reduction. The benefit and utility of Arkade FR reduction over RT-kNN reduction can be clearly seen in the case of L^p norms. With RT-kNN, as p increases, r' increases from r to $\sqrt{3}r$ in 3D space. (The exact value of r' would be $max(r, r \cdot d^{1/2-1/p})$. When p is less than 2, the r' from the inclusion property is the same as the input r.) The side length of AABBs that contain the corresponding spheres also increases from $2r$ to $2\sqrt{3}r$. But with Arkade FR reduction, the side length of AABB remains the same at $2r$. Arkade FR reduction highlights that we do not have to be constricted to using only spheres when we can directly place the distance function geometric objects inside AABB. Moreover, the RT cores index the AABBs and not the spheres.

AABBs in Arkade FR reduction are smaller than in the case of RT-kNN reduction. Tighter and smaller AABBs potentially cause less overlap between AABBs, which in turn reduces the number of unnecessary ray-AABB intersections, thus making the reduction run faster. In L^p norms, as p increases, the geometry of the L^p norm morphs from a sphere into a cube. Note that a cube is also an AABB. Therefore, as p increases, the region inside AABB that does not contribute to the L^p norm geometry decreases. The probability that a ray-AABB intersection would be an output of filter phases increases and reaches 1 for L^∞ norm. Hence, for L^∞ norm, a ray-AABB intersection can be forwarded by the Filter phase of Arkade FR reduction without having to perform an additional geometry check since the geometric object and the AABB are the same. So, Arkade FR reduction achieves optimal performance for a L^∞ norm-based kNN search.

5.2 Other Distances

Jaccard Distance. Jaccard similarity(JS) between two sets A, B is the $A \cap B / A \cup B$ and Jaccard distance is $1 - JS$ [24]. The reduction depends on the type of data and the application for which the data is being ranked. Assume that set A is represented by a bit vector Av where i^{th} bit indicates the presence of i^{th} in the set.

$$JS = \frac{|A \cap B|}{|A \cup B|}$$

Preserving the order according to Jaccard distance and mapping the distance computation is not feasible either using Arkade FR or Arkade MT reductions. With the existing work on repurposing RT architecture, we do not have a way to perform set operations using RT architecture yet. We leave this as a future work.

Hamming Distance. Given two binary data strings, the Hamming distance is the count of bit positions in which the respective bits of the strings are different [18]. Hamming distance is equivalent to the Manhattan distance on binary strings. Indeed, in 3D space, all the possible binary data strings represent vertices of a unit cube, and the Hamming distance between these strings, therefore, is the number of edges that need to be walked from one vertex to the other. Hence, we can use L^1 norm-based Arkade FR reduction.

Mahalanobis Distance. Mahalanobis distance is the distance between a point and a given distribution, where the standard deviation of the point is compared to the mean of the distribution. After a particular spatial transformation, when the axes are scaled to unit variance, Mahalanobis distance is Euclidean distance [27]. Hence, we can use L^2 norm-based Arkade FR reduction.

5.3 Choice of radius

Our reduction require radius (r) as an input parameter. Selecting a *optimal* radius is a challenging task because an arbitrary choice of radius might result in poor performance or accuracy. However, this issue is orthogonal to Arkade's reductions. Presently, we adopt an approach of prior work, TrueKNN [33], which we elaborate on in the Evaluation Section 6. Exploring alternative approaches for determining optimal radius is an intriguing avenue for future investigation.

6 EVALUATION

In this section, we evaluate Arkade's Filter-Refine and Monotone Transformation reductions on four groups of realistic datasets using four baselines. We analyze various factors such as the BVH tree quality, the average number of ray-AABB intersections, and the number of rounds (defined in Sec. 6.2.4) that impact the performance of Arkade on these datasets. Then, we look at the effect of parameters such as k.

Datasets. The characteristics of the datasets we used are summarized in Table 1. As RT cores can only build BVH on three-dimensional data, we use only 2D and 3D datasets. For 2D datasets, we set the third dimension to zero.

Geospatial Datasets Gowalla [8] dataset contains check-in locations of users from across the world in the form of latitude and longitude [8]. We processed the dataset to get only distinct locations. Cali OSM [38] contains geo-spatial coordinates of a very small region in California, sourced from OpenStreetMap. Because the coordinates are local, we treat it as 2-dimensional data. Gbif [12] contains information on several birds and the locations where they are spotted. We obtained the geospatial coordinates of the spottings for January 2018. We convert the geospatial coordinates into Cartesian coordinates before passing the data as input to the Arkade reductions.

Point Clouds Kitti [13] is an autonomous driving footage popularly used in computer vision benchmarks. The data we used is in the form of 3D point clouds generated by the Velodyne scanner. We combined several frames to make

up our dataset. Randnet [7] is a synthetic point cloud generated from real-world and synthetic environments using RandLA-Net architecture [20]. This particular dataset is built on an aerial view of a city landscape.

3D Scans Manuscript [44] dataset is an XYZ RGB 3D scan of a page in Latin from Vellum manuscript.

Synthetic Datasets Glove 3D is a three-dimensional PCA projection of 25-dimensional Glove data [40]. Randnet is also a synthetic dataset.

Table 1: Datasets Characteristics

Dataset	Data Points	Queries	Dimension
Gowalla [8]	1270969	10000	3
Glove 3D [40]	1183514	10000	3
Manuscript [44]	2145617	10000	3
Cali OSM [38]	4195951	10000	2
Kitti [13]	4000000	10000	3
Randnet [20]	6815065	10000	3
Gbif [12]	8475714	10000	3

Baselines. We used three GPU and one state-of-the-art CPU kNN libraries to evaluate Arkade. This mixture contains both tree-based and non-tree-based approaches.

SCANN is a quantization-based *approximate* similarity search library [17]. It is the state-of-the-art in CPU kNN implementations [4]. We use the same parameters as ANN benchmarks [4] to get a recall[2] of 0.99.

Treelogy implements a KD-tree-based *exact* GPU implementation [15]. We modify the Treelogy code to perform L^p and cosine distance-based kNN search.

FAISS is a state-of-the-art *exact* quantization-based GPU library [4, 25]. FAISS uses tensorflow-gpu to interface with CUDA cores. We use the IVFFlatL2 index (as used in ANN benchmarks [4]) and train the data before the search.

FastRNN uses RT architecture to perform fixed-radius search, only in case of Euclidean distances [9]. To correctly perform the kNN search using other distances, we use a larger radius $\sqrt{d}r$, where r is the given radius and d is the data dimension, and a larger number of nearest neighbors k' just enough to obtain k nearest neighbors according to a given distance. (see Subsection 5.1).

We use Treelogy and FastRNN to evaluate the Arkade Filter-Refine reduction, while we use SCANN, FAISS, and Treelogy to evaluate the Arkade Monotone Transformation reduction. SCANN and FAISS implement only L^2 and cosine distances on CPU and GPU respectively. On the other hand, the modifications of FastRNN only work for L^p distances.

Experimental Setup. We used NVIDIA GeForce RTX 4070 Ti GPU with 12GB memory for all of our experiments. To interface with the RT architecture on the GPU, we used Optix Wrapper Library [47]. Arkade builds the BVH tree index once over the entire set of data points for chosen parameters and searches for neighbors once for all the query points in every run. We perform 5 such runs to collect and average the performance metrics such as build time and search time. All the reported numbers are rounded to two non-zero decimals.

We evaluate Filter-Refine reduction with the L^1 and L^∞ distance functions, and Monotone Transformation reduction with cosine distance. We plug in TrueKNN's [33] approach of choosing a small radius and iteratively increasing the radius until all the query points find their k neighbors. To make a fair comparison, we also apply TrueKNN to the baseline FastRNN.

6.1 Performance Evaluation

We compare the search times and the speedups of Arkade reductions over all the baselines and the datasets in Tables 2 and 3. Table 2 shows the comparison of Arkade to the baselines, Treelogy and FastRNN, for L^1 and L^∞ norms. In Table 3, we show the same performance numbers for Cosine distance.

Among all the baselines, we see that Arkade is significantly faster than SCANN, although the speedup can be attributed to SCANN being a purely CPU-based implementation. In the case of GPU baselines, Arkade is still faster by 1.5x-200x. The speedup of Arkade over non-RT baselines demonstrates the ability of RT cores to efficiently accelerate the irregular tree traversals. The speedups over the RT baseline, FastRNN, show how Arkade efficiently utilizes the RT cores to accelerate a broader range of applications.

In general, we find that the speedups of Arkade over baselines do not increase with an increase in the dataset size. For example, Gowalla and Glove3D datasets are roughly 1M in size but Arkade's speedups on these datasets are very different. The search times of non-RT-based implementations such as SCANN, Treelogy, and FAISS increase with the increase in the size of the dataset, however, RT implementations such as Arkade and FastRNN do not follow the same trend. We go into more detail in Section 6.2.1.

6.1.1 L^1 norm. In the first half of Table 2, we see that Arkade achieves speedups of 1.6x-160.9x and 1.3x-33.1x over Treelogy and FastRNN, respectively. Arkade is faster than Treelogy since Arkade uses RT cores to accelerate the BVH tree traversals, while Treelogy uses shader cores.

In this experiment, we use the same search radius for FastRNN and Arkade. This is because the L^1 norm geometric object (rhombus) is present inside the L^2 norm geometric object (circle). Even though the search radius is the same, FastRNN searches for a larger number of neighbors. FastRNN uses L^2 distance to rank the neighbors unlike Arkade, which uses the *actual* distance function, L^1 norm. Because L^1 norm geometric object is smaller in volume compared to L^2 norm geometric object, Arkade can efficiently search neighbors in a smaller space, which is why Arkade is consistently faster than FastRNN.

6.1.2 L^∞ norm. In the second half of Table 2, we see that Arkade achieves speedups of 4.8x-200x and 3.2x-15.6x over Treelogy and FastRNN, respectively. We find that Arkade outperforms Treelogy for the same reason as in the case of the L^1 norm.

As noted in Section 5.1, FastRNN needs a larger search radius ($\sqrt{3}$ times Arkade's radius) and k compared to Arkade. When the radius increases, the size of the AABB increases, which causes an increase in the number of ray-AABB intersection tests performed

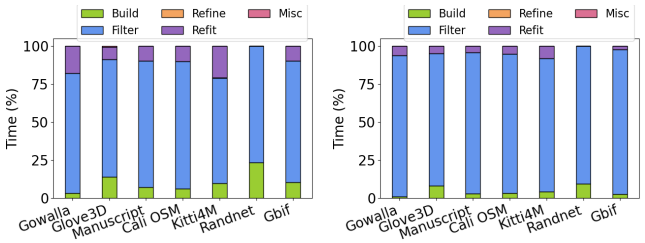
[2]Recall is the ratio of the number of correctly found nearest neighbors by the search to the number of true nearest neighbors from the ground truth.

Table 2: Search times and speedups of Arkade over all baselines for distance functions L^1 and L^∞

Dataset	L^1 distance					L^∞ distance				
	Search time (seconds)			Arkade speedup over		Search time (seconds)			Arkade speedup over	
	Treelogy	FastRNN	Arkade	Treelogy	FastRNN	Treelogy	FastRNN	Arkade	Treelogy	FastRNN
Gowalla	0.16	1.06	0.032	**5.0**	**33.1**	0.16	0.34	0.022	**7.3**	**15.4**
Glove3D	0.16	0.14	0.0063	**25.4**	**22.2**	0.14	0.011	0.0025	**56.0**	**4.4**
Manuscript	0.18	0.075	0.011	**16.4**	**6.8**	0.19	0.032	0.010	**19.0**	**3.2**
CaliOSM	0.25	0.21	0.16	**1.6**	**1.3**	0.24	0.11	0.029	**8.3**	**3.8**
Kitti4M	0.25	0.41	0.045	**5.6**	**9.1**	0.26	0.18	0.043	**6.1**	**4.2**
Randnet	0.37	0.07	0.0023	**160.9**	**30.4**	0.36	0.028	0.0018	**200.0**	**15.6**
Gbif	0.39	1.25	0.082	**4.8**	**15.2**	0.37	1.14	0.077	**4.8**	**14.8**

Table 3: Search times and speedups of Arkade over all baselines for cosine distance function

Dataset	Search time (seconds)				Arkade speedup over		
	SCANN	Treelogy	FAISS	Arkade	SCANN	Treelogy	FAISS
Gowalla	79.37	0.29	0.22	0.10	**793.7**	**2.9**	**2.2**
Glove3D	76.52	0.32	0.21	0.0033	**23,187.9**	**97.0**	**63.6**
Manuscript	149.04	0.47	0.38	0.012	**12,420.0**	**39.1**	**16.7**
CaliOSM	284.67	1.02	0.74	0.013	**21,897.7**	**78.5**	**56.9**
Kitti4M	261.39	0.86	0.7	0.14	**1,867.1**	**6.1**	**5.0**
Randnet	471.12	1.45	1.2	0.026	**18,120.0**	**55.8**	**46.2**
Gbif	581.01	1.81	1.49	0.29	**2,003.5**	**6.2**	**5.1**



(a) Arkade **(b) FastRNN**

Figure 6: Run time breakdown of RT implementations for L^∞ norm-based kNN search

during the BVH traversal. As these intersection tests are the most computationally intensive part of the ray tracing pipeline, we find that Arkade is significantly faster than FastRNN. We further analyze the performance of Arkade and FastRNN in Section 6.2.

6.1.3 Cosine distance. In Table 3, we see that Arkade achieves speedups of 793.7x-23,187.9x, 2.9x-97.0x, and 2.2x-63.6x over SCANN, FAISS, and Treelogy, respectively. Though FAISS is the current state-of-the-art GPU-based kNN search, it is designed for higher dimensional kNN and uses a heavy tensorflow framework. We believe that the combination of FAISS's overheads and Arkade's RT-accelerated neighbor search algorithm results in Arkade's better performance.

6.2 Performance Analysis

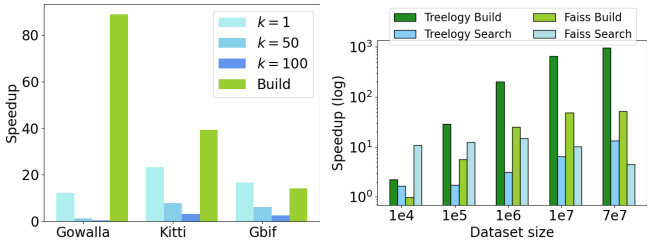
The speedup trend of Arkade can be explained by the data distribution of the dataset. This is because the way the data is distributed affects the quality of the constructed BVH, the number of ray-AABB tests performed for each query point, and, consequently, the number of candidates the Filter phase forwards to the Refine phase. We unroll the effects of data distribution on each of the reductions in the following subsections.

6.2.1 Breakdown. To understand the factors impacting the Arkade speedups, we present a complete breakdown of Arkade and FastRNN execution times for L^∞ norm in Figure 6. The execution time is comprised of both BVH build and search times. The search time is further divided into four parts – time taken by the Filter phase, Refine phase, refit, and miscellaneous maintenance in between these steps.

Figures 6a and 6b show the breakdown of execution times of Arkade and FastRNN for L^∞ distance, respectively. The percentage of build time is higher in the case of Arkade than in FastRNN. However, the actual build times in both cases are approximately the same for respective datasets. Because Arkade's search times are lower than FastRNN's, the percentage of build time of Arkade is higher.

In Figures 6a and 6b, the Filter phase predominantly takes more time than any other steps. In the Filter phase, the ray traverses the BVH and checks if it intersects an AABB, and when it does intersect an AABB, it further checks if the ray intersects the geometry. The time the filter phase takes is affected by the quality of BVH the RT architecture constructs. The structure of BVH further impacts the BVH traversal and the number of intersection tests performed.

6.2.2 Impact of BVH Tree ality. The negligible amount of time spent in the Refine phase (Refine time is barely visible in Figure 6)



(a) For $k = 1, 50, 100$, Arkade's search and build time speedup over Faiss

(b) Arkade's search and build time speedups over Faiss and Treelogy as the dataset size changes

Figure 7: Sensitive analysis of Arkade's search and build time speedups for Cosine distance

supports the observation that more time is spent in traversal and filtering AABBs rather than ordering the candidates present inside them. The mapping of the kNN problem to RT architecture needs the geometric objects to overlap to produce results. As the k value increases, the search radius needed to find all k neighbors also increases. This increases the potential of geometric objects to overlap and reduces the effectiveness of the BVH in pruning large parts of the neighbor search space, resulting in a BVH of poor quality. High overlap, in turn, increases the number of ray-AABB intersections.

6.2.3 Impact of ray-AABB intersections. In Table 4, we present the average number of ray-AABB intersections per query point that occurred in the RT-based implementations in the case of L^∞ and Cosine distances. In the case of Cosine distance, the search times of Arkade increase with an increase in the number of intersections and decrease with a decrease in the number of intersections. Similarly, in the case of L^∞ distance, Arkade and FastRNN search times are proportional to the number of intersections except for the Kitti4M dataset. Moreover, the number of intersections is higher for any dataset in the case of Cosine distance compared to L^∞ norm, and we observe that Cosine distance-based search takes longer than that of L^∞ norm. Due to normalization, the points become denser in the Cosine distance scenario.

6.2.4 Impact of number of rounds. While the number of intersections explains most of the trends in search times of RT-based implementations, there are certain instances where the number of intersections alone does not suffice. For example, FastRNN spends more search time on Kitti4M than the Randnet dataset, but the number of ray-AABB intersections on Kitti4M is lower than that of Randnet. We observe that the number of rounds is higher in the case of Kitti4M than in Randnet. We present the number of rounds for each RT-implementation and dataset in Table 4. The number of rounds is the number of times TrueKNN doubles the radius until it finds k nearest neighbors of all query points.

A higher number of rounds increases the refit time. In Figure 6, we see that refit is the second most time-consuming part of the search. The refit time corresponds to doubling the radius of geometries, updating the AABBs to fit the new larger geometries, and refitting the BVH for every round.

The need for a higher number of rounds arises from the data distribution. A new round is performed when the neighbors of some of the query points can only be found at a larger radius. Hence, the number of rounds indicates that some neighborhoods of the dataset are denser than others.

6.2.5 Sensitivity to k. In Figure 7a, we study the speedup of Arkade performance over FAISS in the case of Cosine distance as k increases. We vary from k as 1, 50, and 100 on Gowalla, Kitti4M, and Gbif datasets. We also plot the build time speedup of Arkade over FAISS for each dataset. Arkade's search time speedups decrease as k increases. But, observe that the build times of Arkade are much lower than FAISS. So the overall runtime (build + search time) of Arkade is still lower than FAISS. At a sufficiently large dataset size and k, it is possible that the benefit of using Arkade might diminish. However, in practice, k is typically at most 100 [4].

6.2.6 Sensitivity to Dataset size. In Figure 7b, we study the speedup (in log scale) of Arkade performance over Treelogy and FAISS in the case of Cosine distance as the magnitude of the dataset size increases. We uniformly sample a randomly generated dataset to get the number of data points from 10K to 70M. In Figure 7b, green shaded bars show the speedup of Arkade build times over Treelogy and Faiss, respectively, while blue shaded bars show the speedup of Arkade search times over the same baselines. Arkade's build and search time speedups over Treelogy increase almost linearly with the increasing magnitude of the dataset size. We attribute these speedups to the optimized strategies employed for tree construction and traversal by the RT architecture. Conversely, search time speedups over Faiss slightly decrease when the dataset size reaches 10M. We also note that Arkade runs out of memory on our 12GB GPU after 70M points, however, both the baselines can execute up to 100M points.

6.2.7 Impact of hardware utilization. Available Nvidia profilers [36] can not differentiate RT cores from shader cores. Nvidia 4060Ti, the GPU on which we ran our experiments, has 4352 and 32 shader and RT cores, respectively. Without knowing how the architecture maps the point ray queries to the hardware, it is difficult to determine if Arkade is saturating the resources on RT architecture. However, applying optimizations like balancing the workload among the threads may improve the utilization and performance. We leave this for future work.

7 RELATED WORK

7.1 Non-RT Applications Accelerated With RT Architecture

Recent work has shown that non-ray-tracing problems can be expressed as ray-object intersection problems, making them amenable to acceleration with RT cores [9, 32, 48, 52, 54]. Wald *et. al.* [48] were the first to use RT cores to accelerate non-ray tracing applications. They looked at the problem of identifying the location of a point in a tetrahedral mesh. By modeling the point as a ray and reporting the closest tetrahedron intersected by the ray, they identified the tetrahedron in which the point was contained. Zellman *et. al.* [52] showed how to use RT cores to perform graph drawing. They reformulated the nearest neighbor search subroutine as a ray tracing problem and used the force exerted by the nearest neighbors to direct their graph drawing algorithm. They found their approach to be significantly faster than the state-of-the-art force-directed

Table 4: Average number of ray-AABB intersection and number of rounds for L^∞ distance (Table 2) and cosine distance (Table 3).

Dataset	L^∞ distance				Cosine distance	
	Arkade		FastRNN		Arkade	
	Average #Intersections	Rounds	Average #Intersections	Rounds	Average #Intersections	Rounds
Gowalla	263.47	10	510.40	10	10613.80	7
Glove3D	26.12	2	60.46	2	60.46	2
Manuscript	173.20	4	510.50	4	357.77	3
CaliOSM	440.11	6	827.03	6	2695.24	1
Kitti4M	366.38	8	365.28	7	20669.20	1
Randnet	121.92	1	397.79	1	211.86	4
Gbif	3093.40	5	5185.58	5	20708.30	1

graph drawing algorithms. Evangelou *et. al.* [9] used RT cores to perform photon mapping by finding the set points in a fixed-radius neighborhood of a query point. They used the reduction proposed by Zellman *et. al.* and found that they were up to 15x faster than non-RT-accelerated baselines. Zhu *et. al.* [54] proposed optimizations such as point reordering and query partitioning to improve the performance of RT-accelerated neighbor searches. Nagarajan *et. al.* proposed RT-DBSCAN [32] and TrueKNN [33] to leverage RT cores to solve DBSCAN clustering and efficiently perform k-nearest neighbor search, respectively.

7.2 Tree-based, GPU-accelerated kNN

Tree-based kNN algorithms are only efficient at lower dimensions due to the curse of dimensionality [49]. They are mostly specialized for certain applications. Merry *et. al.* [29] propose an optimization to leverage the coherence of points when traversed in kd tree order so as to reuse traversal information of neighboring points. They find that their approach is 4.4x to 4.6x faster and does not require any modifications to the kd tree. Treelogy[15, 19] proposes several optimizations to improve memory coalescing and reduce divergence caused by GPU threads that traverse different parts of the tree. Gieseke *et. al.* [14] propose the idea of a buffer kd tree to create batches of query points that all target the same leaf nodes of the kd tree, exploiting data locality. However, their work is specialized for data with dimensionality between 4 and 25. An optimized approximate KD-tree-based KNN is proposed to aid in point cloud registration [51]. However, this optimization is application-specific. Gowanlock [16] proposes a hybrid CPU-GPU algorithm that breaks computation up so that areas of large density are assigned to the GPU, while the CPU handles the rest of the data. This approach leverages the advantages offered by the different architectures to optimize performance.

8 CONCLUSION

Irregular problems like tree traversals are ubiquitous, especially queries like nearest neighbor search that have applications in domains such as point cloud registration in computer vision, data compression, similarity scoring, DNA sequencing, etc. Tree-based nearest neighbor search is naturally challenging to scale up using purely software approaches on massively parallel commodity hardware such as GPUs. Even though ray tracing cores of GPU are specialized hardware to cater to graphics applications, we show that this specialized hardware can be generalized to accelerate tree operations in other domains, To that end, we provide a set of reductions to the ray tracing scene. Without our reductions, distance metric computations such as L^p norm and cosine distance take significantly longer to complete or cannot be run on RT cores (it varies between previous works). While RT cores accelerate tree traversals through BVH construction, this tree structure is not accessible to the user and is limited to 3D space. Availability and programmability of the spatial tree itself would be more helpful in using RT cores for general applications.

ACKNOWLEDGMENTS

We are thankful to all the anonymous reviewers for providing valuable feedback. We also thank Kirshanthan Sundararajah for helping us improve the earlier versions of the paper and Raghav Malik for helping us with the proof. This work was funded by NSF grants CCF-1908504, CCF-1919197 and CCF-2216978.

REFERENCES

[1] David Adedayo Adeniyi, Zhaoqiang Wei, and Yang Yongquan. 2016. Automated web usage data mining and recommendation system using K-Nearest Neighbor (KNN) classification method. *Applied Computing and Informatics* 12, 1 (2016), 90–108.
[2] AMD. 2023. AMD Ray tracing. https://www.amd.com/en/technologies/rdna
[3] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. 2015. Cache locality is not enough: High-Performance Nearest Neighbor Search with Product Quantization Fast Scan. *Proc. VLDB Endow.* 9, 4 (2015), 288–299.
[4] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems* 87 (2020), 101374. https://doi.org/10.1016/j.is.2019.02.006
[5] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (1975), 509–517.
[6] N. Bourbaki. 1987. *Topological Vector Spaces: Chapters 1-5.* Springer-Verlag, Berlin. https://books.google.com/books?id=S4wnAQAAIAAJ
[7] Meida Chen, Qingyong Hu, Zifan Yu, Hugues THOMAS, Andrew Feng, Yu Hou, Kyle McCullough, Fengbo Ren, and Lucio Soibelman. 2022. STPLS3D: A Large-Scale Synthetic and Real Aerial Photogrammetry 3D Point Cloud Dataset. In *33rd British Machine Vision Conference, November 21-24, 2022.* BMVA Press, London, UK, 429. https://bmvc2022.mpi-inf.mpg.de/0429.pdf
[8] E. Cho, S. A. Myers, and J. Leskoven. 2023. Friendship and Mobility: User Movement in Location-Based Social Networks. Retrieved from UCR-STAR https://star.cs.ucr.edu/?stanford-gowalla&d.
[9] I. Evangelou, G. Papaioannou, K. Vardis, and A. A. Vasilakis. 2021. Fast Radius Search Exploiting Ray Tracing Frameworks. *Journal of Computer Graphics Techniques (JCGT)* 10, 1 (5 February 2021), 25–48. http://jcgt.org/published/0010/01/02/

[10] Jerome H Friedman, Jon Louis Bentley, and Raphael Ari Finkel. 1977. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)* 3, 3 (1977), 209–226.

[11] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graph. *Proc. VLDB Endow.* 12, 5 (2019), 461–474.

[12] GBIF.Org User. 2023. Occurrence Download. https://doi.org/10.15468/DL.QQ7KRQ

[13] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. 2013. Vision meets Robotics: The KITTI Dataset. https://www.cvlibs.net/datasets/kitti/raw_data.php

[14] Fabian Gieseke, Justin Heinermann, Cosmin E. Oancea, and Christian Igel. 2014. Buffer k-d Trees: Processing Massive Nearest Neighbor Queries on GPUs. In *ICML (JMLR Workshop and Conference Proceedings, Vol. 32)*. JMLR.org, 172–180.

[15] Michael Goldfarb, Youngjoon Jo, and Milind Kulkarni. 2013. General Transformations for GPU Execution of Tree Traversals. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (Denver, Colorado) *(SC '13)*. Association for Computing Machinery, New York, NY, USA, Article 10, 12 pages. https://doi.org/10.1145/2503210.2503223

[16] Michael Gowanlock. 2021. Hybrid KNN-join: Parallel nearest neighbor searches exploiting CPU and GPU architectural features. *J. Parallel and Distrib. Comput.* 149 (2021), 119–137.

[17] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2019. Accelerating Large-Scale Inference with Anisotropic Vector Quantization. https://doi.org/10.48550/ARXIV.1908.10396

[18] R. W. Hamming. 1950. Error detecting and error correcting codes. *The Bell System Technical Journal* 29, 2 (1950), 147–160. https://doi.org/10.1002/j.1538-7305.1950.tb00463.x

[19] Nikhil Hegde, Jianqiao Liu, Kirshanthan Sundararajah, and Milind Kulkarni. 2017. Treelogy: A benchmark suite for tree traversals. In *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 227–238. https://doi.org/10.1109/ISPASS.2017.7975294

[20] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. 2020. RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2020).

[21] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *Proceedings of the VLDB Endowment* 9, 1 (2015), 1–12.

[22] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 604–613.

[23] Intel. 2023. Intel Ray tracing. https://www.intel.com/content/www/us/en/developer/articles/guide/real-time-ray-tracing-in-games.html

[24] Paul Jaccard. 1912. THE DISTRIBUTION OF THE FLORA IN THE ALPINE ZONE.1. *New Phytologist* 11, 2 (1912), 37–50. https://doi.org/10.1111/j.1469-8137.1912.tb05611.x

[25] J. Johnson, M. Douze, and H. Jegou. 2021. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data* 7, 03 (Jul 2021), 535–547. https://doi.org/10.1109/TBDATA.2019.2921572

[26] Lukasz Kaiser and Ilya Sutskever. 2015. Neural GPUs Learn Algorithms. https://doi.org/10.48550/ARXIV.1511.08228

[27] Prasanta Chandra Mahalanobis. 1936. On the generalised distance in statistics. http://library.isical.ac.in:8080/xmlui/bitstream/handle/10263/6765/Vol02_1936_1_Art05-pcm.pdf

[28] Yu A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 4 (apr 2020), 824–836. https://doi.org/10.1109/TPAMI.2018.2889473

[29] Bruce Merry, James Gain, and Patrick Marais. 2013. Accelerating kd-tree Searches for all k-nearest Neighbours. In *Eurographics 2013 - Short Papers*, M.-A. Otaduy and O. Sorkine (Eds.). The Eurographics Association. https://doi.org/10.2312/conf/EG2013/short/037-040

[30] G. Mori, S. Belongie, and J. Malik. 2001. Shape contexts enable efficient retrieval of similar shapes. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, Vol. 1. I–I. https://doi.org/10.1109/CVPR.2001.990547

[31] Marius Muja and David G. Lowe. 2014. Scalable Nearest Neighbor Algorithms for High Dimensional Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36, 11 (2014), 2227–2240. https://doi.org/10.1109/TPAMI.2014.2321376

[32] Vani Nagarajan and Milind Kulkarni. 2023. RT-DBSCAN: Accelerating DBSCAN using Ray Tracing Hardware. In *IPDPS*. IEEE, 963–973.

[33] Vani Nagarajan, Durga Mandarapu, and Milind Kulkarni. 2023. RT-kNNS Unbound: Using RT Cores to Accelerate Unrestricted Neighbor Search. In *Proceedings of the 37th International Conference on Supercomputing, ICS 2023, Orlando, FL, USA, June 21-23, 2023*, Kyle A. Gallivan, Efstratios Gallopoulos, Dimitrios S. Nikolopoulos, and Ramón Beivide (Eds.). ACM, 289–300. https://doi.org/10.1145/3577193.3593738

[34] Moohyeon Nam, Jinwoong Kim, and Beomseok Nam. 2016. Parallel tree traversal for nearest neighbor query on the GPU. In *2016 45th International Conference on Parallel Processing (ICPP)*. IEEE, 113–122.

[35] NVIDIA. [n. d.]. NVIDIA OptiX 7.5 – Programming Guide. https://raytracing-docs.nvidia.com/optix7/guide/index.html

[36] Nvidia. 2023. NVIDIA Nsight Compute. https://developer.nvidia.com/nsight-compute

[37] Nvidia. 2023. NVIDIA Ray tracing. https://developer.nvidia.com/rtx/ray-tracing

[38] OpenStreetMap. [n. d.]. https://www.openstreetmap.org

[39] Varun Pandey, Andreas Kipf, Thomas Neumann, and Alfons Kemper. 2018. How good are modern spatial analytics systems? *Proceedings of the VLDB Endowment* 11, 11 (2018), 1661–1673.

[40] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. 1532–1543. http://www.aclweb.org/anthology/D14-1162

[41] Ninh Pham and Tao Liu. 2022. Falconn++: A Locality-sensitive Filtering Approach for Approximate Nearest Neighbor Search. arXiv:2206.01382 [cs.DS]

[42] Deyuan Qiu, Stefan May, and Andreas Nüchter. 2009. GPU-accelerated nearest neighbor search for 3D registration. In *Computer Vision Systems: 7th International Conference on Computer Vision Systems, ICVS 2009 Liège, Belgium, October 13-15, 2009. Proceedings 7*. Springer, 194–203.

[43] Mark J. Reid and Karl M. Menten. 2020. The first stellar parallaxes revisited. *Astronomische Nachrichten* 341, 9 (nov 2020), 860–869. https://doi.org/10.1002/asna.202013833

[44] The Stanford 3D Scanning Repository. 2014. Vellum manuscript, The XYZ RGB models. http://graphics.stanford.edu/data/3Dscanrep/

[45] Steven Rubin and Turner Whitted. 1980. A 3-dimensional representation for fast rendering of complex scenes. *ACM Siggraph Computer Graphics* 14. https://doi.org/10.1145/965105.807479

[46] Amit Singhal. 2001. Modern Information Retrieval: A Brief Overview. *IEEE Data Eng. Bull.* 24, 4 (2001), 35–43. http://sites.computer.org/debull/A01DEC-CD.pdf

[47] Ingo Wald, Nathan Morrical, and Haines E. [n. d.]. OWL: A Node Graph "Wrapper" Library for OptiX 7. https://github.com/owl-project/owl

[48] Ingo Wald, Will Usher, Nathan Morrical, Laura Lediaev, and Valerio Pascucci. 2019. RTX Beyond Ray Tracing: Exploring the Use of Hardware Ray Tracing Cores for Tet-Mesh Point Location. In *High-Performance Graphics - Short Papers*, Markus Steinberger and Tim Foley (Eds.). The Eurographics Association. https://doi.org/10.2312/hpg.20191189

[49] Roger Weber, Hans-Jörg Schek, and Stephen Blott. 1998. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *VLDB*.

[50] Jordan Wood. 2008. *Filter and Refine Strategy*. Springer US, Boston, MA, 320–320. https://doi.org/10.1007/978-0-387-35973-1_415

[51] Tiancheng Xu, Boyuan Tian, and Yuhao Zhu. 2019. Tigris: Architecture and Algorithms for 3D Perception in Point Clouds. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (Columbus, OH, USA) *(MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 629–642. https://doi.org/10.1145/3352460.3358259

[52] Stefan Zellmann, Martin Weier, and Ingo Wald. 2020. Accelerating Force-Directed Graph Drawing with RT Cores. In *2020 IEEE Visualization Conference (VIS)*. 96–100. https://doi.org/10.1109/VIS47514.2020.00026

[53] Weijie Zhao, Shulong Tan, and Ping Li. 2020. SONG: Approximate Nearest Neighbor Search on GPU. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 1033–1044. https://doi.org/10.1109/ICDE48307.2020.00094

[54] Yuhao Zhu. 2022. RTNN: Accelerating Neighbor Search Using Hardware Ray Tracing. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (Seoul, Republic of Korea) *(PPoPP '22)*. Association for Computing Machinery, New York, NY, USA, 76–89. https://doi.org/10.1145/3503221.3508409