X-HD: Fast Hausdorff Distance Computation with Ray Tracing

Liang Geng
The Ohio State University
Columbus, Ohio, USA
geng.161@osu.edu

Zhehu Yuan
The Ohio State University
Columbus, Ohio, USA
yuan.645@osu.edu

Rubao Lee
The Ohio State University
Columbus, Ohio, USA
lee.11875@osu.edu

Fusheng Wang
Stony Brook University
Stony Brook, New York, USA
fusheng.wang@stonybrook.edu

Xiaodong Zhang
The Ohio State University
Columbus, Ohio, USA
zhang.574@osu.edu

Abstract

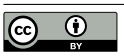
The Hausdorff distance (HD) is a mathematical measure of the similarity (or dissimilarity) between two sets of points, typically used to compare geometric shapes, images, or spatial distributions. HD computation has a broad range of applications in large-scale data analysis across domains like medical imaging, geospatial information systems, and graphics. As the volume of spatial data continues to grow rapidly, HD computation has become increasingly time-consuming, demanding effective and scalable hardware acceleration. Recent research on utilizing Ray Tracing (RT) cores to accelerate k-nearest neighbor (k-NN) search shed light on this problem, as k-NN is the building block of the HD algorithm. However, naively using an RT-accelerated k-NN library yields poor performance due to the lack of domain-specific optimizations, leading to poor utilization of hardware resources. In this paper, we propose *X-HD*, a general-purpose HD algorithm with RT acceleration for large-scale datasets. *X-HD* has three core optimization techniques: (1) We use a grid to organize spatially proximal points, which reduces the traversal intensity of the Bounding Volume Hierarchy (BVH) tree. (2) We use HD estimators to prune non-contributing points for faster processing. (3) Introducing the grid also results in severe load imbalance in RT shaders. Our solution addresses this by selectively offloading the distance computation from RT shaders to a dedicated CUDA kernel to improve load balance. *X-HD* outperforms the state-of-the-art industrial software *ITK* by 5.3× on average. Compared to a GPU-optimized HD solution, *X-HD* achieves a speedup of up to 6.4×.

CCS Concepts

• **Computing methodologies** → **Massively parallel algorithms**; **Ray tracing**; • **Information systems** → *Spatial-temporal systems.*

Keywords

Hausdorff distance, ray tracing, GPU acceleration



ICS '26, Belfast, United Kingdom

ACM ISBN 979-8-4007-2522-7/26/07
https://doi.org/10.1145/3797905.3800509

ACM Reference Format:
Liang Geng, Zhehu Yuan, Rubao Lee, Fusheng Wang, and Xiaodong Zhang. 2026. X-HD: Fast Hausdorff Distance Computation with Ray Tracing. In *2026 International Conference on Supercomputing (ICS '26), July 06–09, 2026, Belfast, United Kingdom.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3797905.3800509

1 Introduction

The Hausdorff Distance (HD) is a spatial metric used to measure the dissimilarity between two sets of points. It quantifies the maximum distance from a point in one set to the nearest point in the other set, effectively capturing the greatest discrepancy between the two. HD is an important tool in various fields, including CAD [9, 42, 46, 75, 79], image processing [4, 11, 31, 35, 54, 68], computational geometry [2, 30], and spatial databases, such as PostgreSQL [52, 63].

The volume of spatial data has expanded rapidly in recent years, driven by the widespread use of high-resolution sensors, such as LiDAR, GPS-enabled devices, satellite imagery, and advanced medical imaging technologies. For example, datasets collected from LiDAR sensors to train self-driving cars can contain billions of points [7]. At the same time, spatial workloads are often time-sensitive [71], making it imperative to develop super fast HD algorithms supported by an effective hardware acceleration.

The HD computation is highly intensive and time-consuming. Existing approaches to solving HD typically follow its definition, where each point in one set finds its nearest point in the other set. Subsequently, the HD can be computed by taking the maximum of the distances between the points and their closest counterparts. The above operation is a special case of k-nearest neighbor (k-NN) search where $k = 1$. Using a tree-like data structure, e.g., KD-tree, to solve the problem on the GPU can be very slow due to its inherent recursive nature, which causes excessive branch divergence and random memory access [32, 83].

Recent research has shown that using Ray-Tracing (RT) cores is beneficial for solving the k-NN problem [47, 56, 83]. RT cores are designed to accelerate photorealistic rendering, which are ubiquitous in consumer-grade GPUs, data centers, and supercomputing environments [1, 8]. To utilize these specialized units, the k-NN search must be reformulated as a ray-tracing problem, such as searching ray-Axis-Aligned Bounding-Box (AABB) intersections

by traversing a Bounding Volume Hierarchy (BVH) tree, and the tree traversal is conducted on the RT cores.

However, naively invoking an RT-based k-NN library to solve HD yields poor performance. In existing works, such as *RTNN* [83] and *RT-KNNS Unbound* [56], each point is mapped to an individual AABB to exploit the RT-accelerated BVH search. This strategy leads to significant BVH traversal and intersection test overhead due to the large volume of points. In addition, some crucial domain-specific optimizations, such as Early Break (EB) [67] and HD estimators [57], cannot be applied in these libraries to reduce redundant computation, as they are viable only for the HD algorithm. Finally, the existing RT programming model, e.g., NVIDIA OptiX [58], lacks a load-balancing mechanism in the RT shader, leading to poor hardware resource utilization.

To address the above mentioned issues, we propose the following design principles for a high-performance HD algorithm on RT cores.

(1) We aim to reduce the intensity of the BVH traversal and intersections. Our measurement shows that handling an intersection is about 5× more expensive than calculating the distance between two points.
(2) The intensity of distance computation should be significantly reduced because the value of HD only comes from a single pair of points, implicating many non-contributing points can be culled.
(3) Load-balancing should be carefully considered because RT cores adopt a single-ray programming model, and poor load-balancing will significantly underutilize the GPU [58].

In this paper, we introduce *X-HD*, an RT-accelerated HD algorithm designed for high-performance processing of large-scale datasets. *X-HD* has three core techniques. First, we use a uniform grid to group spatially proximal points into a cell, which is inexpensive to build. Subsequently, each non-empty cell is extended by a search radius to create an AABB, which serves as the input for building the BVH. This step not only reduces the BVH size for faster construction but also results in fewer intersections and better spatial locality [14]. Reducing non-contributing points is our second core technique. Using the cell boundaries, we employ HD estimators to calculate the lower and upper bounds of HD, which helps us to prune non-contributing points within the cell. *X-HD* also incorporates the EB optimization to cull individual points that cannot increase the value of HD. Finally, while introducing a grid offers benefits, it also brings new issues, such as imbalanced workloads due to skewed data distribution. For better load balancing, *X-HD* selectively offloads the distance computation from the RT shader to a dedicated CUDA kernel to improve load balance.

Our experiments show that *X-HD* outperforms the state-of-the-art medical imaging framework *ITK* by 5.3× on average with MRI images. Compared to a GPU-optimized EB algorithm, *X-HD* achieves a speedup of up to 6.4× on large-scale geospatial datasets.

The contributions of this paper are as follows:

(1) We designed and implemented a highly optimized HD algorithm tailored for execution on RT cores.
(2) We proposed using grid and HD estimators to reduce the intensity of BVH traversal and distance computation. The latter is beyond HD - it can also be incorporated into RT-based k-NN solutions [60].
(3) We investigated the load balance issues inherent in RT shaders and proposed a general solution to improve load balance. This technique can be used by any RT-accelerated algorithms facing load-balancing challenges.
(4) We have comprehensively compared and analyzed *X-HD* with five baselines using diverse real-world datasets.
(5) Putting all the design and implementation efforts together, we created *X-HD* as an open-source software for the research and application communities [1].

2 Background

2.1 Hausdorff Distance

Given two sets of points P^A and P^B, where $P^A = \{p_1^a, p_2^a, \ldots, p_m^a\}$ and $P^B = \{p_1^b, p_2^b, \ldots, p_n^b\}$ in a metric space with distance function $\|\cdot\|$. The directed Hausdorff distance from P^A to P^B is:

$$\overrightarrow{H}(P^A, P^B) = \max_{p^a \in P^A} \{ \min_{p^b \in P^B} \{\|p^a, p^b\|\}\} \quad (1)$$

Note that the directed Hausdorff distance is not symmetric, e.g., $\overrightarrow{H}(P^A, P^B) \neq \overrightarrow{H}(P^B, P^A)$. The undirected Hausdorff distance can be trivially calculated by taking the maximum of the two directed Hausdorff distances, which yields a symmetric measure. While the directed HD finds its applications in graphics, such as finding the best partial distance between two images [31], the undirected HD can be applied to evaluate the quality of a medical image segmentation algorithm [33] and trajectory analysis [61]. In this paper, we focus on the directed Hausdorff distance for point sets in 2-D/3-D Euclidean space as it has wide applications [24, 31, 57, 66].

2.2 HD Algorithms

To compute HD, we can directly follow its definition in Equation 1 with a double-loop scheme. The outer loop iterates over every point $p^a \in P^A$, while the inner loop finds the minimum distance, *cmin*, from p^a to the points in P^B. After each inner loop, the global maximum distance, *cmax*, is updated with $cmax = max(cmax, cmin)$ to maintain the maximum of the minimum distances found. The time complexity[2] of this algorithm is $O(|P^A| \cdot |P^B|)$, which is not scalable and time-consuming for large datasets.

Early Break There is an opportunity to optimize the double loop by skipping points that cannot contribute to the Hausdorff distance, achieved by breaking the inner loop earlier [67]. As shown in Algorithm 1, the distance of a pair of points, *dist*, is checked against *cmax*. If *dist* is not greater than *cmax*, we can safely exit the inner loop. This pruning logic can be interpreted as follows: if the current minimum distance (*cmin*) found for point p^a is already less than *cmax*, then continuing the inner loop to yield a lower distance still can not be greater than *cmax* and thus cannot further increase the overall Hausdorff distance *cmax*.

This algorithm can be very efficiently implemented on the GPU. For the outer loop, we dispatch a thread block to take point p^A, while the inner loop can be unrolled with the thread block. In Line 8, we use a block-level reduction to aggregate the minimum distance

[1] Source code: https://github.com/pwrliang/X-HD
[2] We consider the spatial dimension as a small constant factor, e.g., 2 or 3.

from each thread and vote to break the loop with a thread block synchronization.

Algorithm 1: Hausdorff Distance with Early Break

```
Input  : Point Set P^A
Input  : Point Set P^B
Output : Hausdorff Distance
Shuffle(P^A) // Randomize points to avoid similar distances in
    successive iterations [67]
Shuffle(P^B)
cmax ← 0
for each p^a in P^A do
    cmin ← ∞      // Min distance discovered from p^a to P^B
    for each p^b in P^B do
        dist ← ||p^a, p^b||
        cmin ← min(cmin, dist)
        if dist ≤ cmax then
            break // p^A cannot further increase cmax
        end if
    end for
    cmax ← max(cmax, cmin)
end for
return cmax
```

Nearest-Neighbor Search Since the inner loop in Algorithm 1 essentially computes the distance from p^a to its nearest neighbor point, we can avoid visiting all the points in P^B by formulating the inner loop as a k-NN problem, where $k = 1$.

We first build a spatial index, e.g., a KD-tree, over the point set P^B. Then, we iterate over the point set P^A, performing a nearest neighbor search for each point $p^a \in P^A$ using the spatial index. Next, we calculate the distance between p^a and its nearest neighbor to update $cmax$. With the index, the average time complexity is $O(|P^A| \cdot \log|P^B|)$.

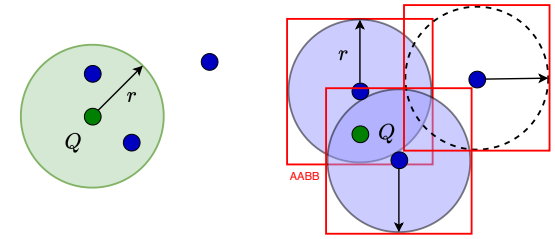
Although introducing a spatial index will reduce the work amount, resulting fewer distance computations, tree-like data structures are not efficient for the GPU due to the random memory accesses and branch divergence [83]. Our experiments show that using a KD-tree is almost always slower than the EB solution (§6.2). In this paper, we will show how we turn this hardware inefficient algorithm into a GPU-friendly solution with RT cores.

2.3 RT-accelerated Nearest Neighbor Search

The basic idea of RT-based NN search originates from an idea known as the "inverse test" [20, 83]. For instance, a query point Q seeks to identify its nearest neighbor within a specified radius r (Fig. 1(a)). This problem can be equivalently reformulated as an inverse test. As shown in Fig. 1(b), the problem is reversed to ask: "Which spheres of radius r, centered at data points, contain the query point Q?"

By utilizing this inverse test approach, we employ an AABB to enclose each sphere and cast rays from the query points to find potential intersections. The AABBs are defined by expanding the points by a radius r. A query point finds its nearest neighbor if its ray hits an AABB and the distance to the AABB's center is no more than r[3]. Otherwise, the search continues, and the point is reprocessed with an increased r (§5).

[3]In cases of multiple AABB intersections, the shortest distance is recorded.



(a) Neighbor search for query Q with radius r (b) Neighbor search with inverse test

Figure 1: (a) Query Q searches its nearest neighbor within radius r; (b) Achieve the query with the inverse test that finds ray-AABB intersections followed by a point-in-sphere test

Searching for these intersections can be achieved by traversing the BVH built using the AABBs, and the traversal is automatically accelerated by RT cores. This design is therefore significantly faster than software-emulated tree search solutions [83]. The existing RT-accelerated NN search algorithms are based on NVIDIA's OptiX framework because of its support for BVH traversal acceleration. To write an RT program, developers must write RT shaders — essentially a piece of CUDA code, but executed from the perspective of a single thread. When the RT cores report a potential intersection, an RT shader called `IsIntersection` is invoked, allowing the developer to handle the intersection event. For the NN search case, the `IsIntersection` shader calculates the distance between the ray origin and the center of the AABB being hit and maintains the minimum distance discovered so far. Once the shader completes, the execution flow transfers from the CUDA cores back to the RT cores to continue the search process.

3 Design Principles

Directly using the RT-based NN algorithms to solve HD results in very poor performance due to the cost of building a large BVH, redundant computation, and load imbalance. We discuss these equally important issues and the ideas to solve them in this section.

3.1 Reducing BVH Size

The existing RT-based k-NN algorithms, such as *RTNN* [83] and *RT-KNNS Unbound* [56], adopt a one-to-one mapping between a point and an AABB (§2.3). This simple mapping scheme results in a high BVH construction cost and reduces the efficiency of filtering intersections. Existing research has shown that the building time is linearly correlated with the number of AABBs [83]. When the number of points is high, BVH construction becomes a performance bottleneck. A large BVH also results in more intersections. When casting a ray to search the BVH, whenever a leaf node is visited, `IsIntersection` will be invoked to notify the users to handle potential ray-AABB intersections. The more AABBs a BVH has, the more AABBs a ray will possibly hit.

To reduce the BVH size, we should allow an AABB to enclose multiple points. Only spatially proximal points should be grouped to reduce the dead space[4] of the AABB, leading to fewer false-positive intersections. If an AABB is hit, we can iterate over the points in the cell and narrow down the nearest point. There are many data

[4]The dead space refers to the area in the AABB that has no geometries [19]

structures that serve the above purposes. For example, a quad-tree or KD-tree can be used to partition the space, and spatially proximal points will be naturally placed in a tree node. However, we have tried them and found that these complex data structures are very expensive to build on the GPU due to their recursive nature.

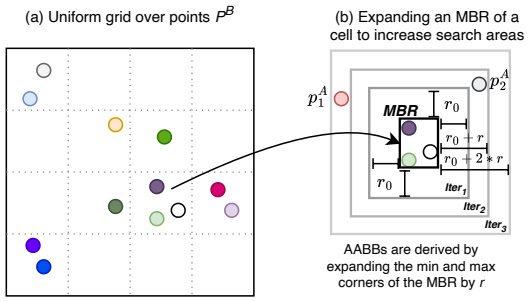


Figure 2: (a) using a uniform grid to organize spatially proximal points; (b) Create an MBR for points in a cell and iteratively expand the search radius.

A uniform grid [23] is a simple and widely used data structure [34, 74] that partitions the space into many identical, square-sized cells, as shown in Fig. 2(a). With a predefined grid, a point can be mapped into its cell with $O(1)$ time complexity, and the storage for the grid can be a simple array. Using the grid, proximal points are automatically placed into the same cell. Introducing a grid also causes other issues, such as load imbalance and determining an appropriate grid resolution.

3.2 Reducing Redundant Computation

3.2.1 EB and NN are not Mutually Exclusive. Recall that EB is essentially a pruning technique that skips a non-contributing point in P^A. While the NN method adopts a spatial index to prune the points in P^B that are very far from P^A. In short, these two methods prune non-contributing points from P^A and P^B, respectively.

These two methods can be used together. We could adopt the NN-based algorithm as the framework while incorporating the EB technique simultaneously. In the process of the NN search, if the current minimum distance (*cmin*) found for point p^A is lower than the global maximum distance (*cmax*), it is safe to exit from the search process early. Compared to the EB algorithm, which iterates through every point in P^B, using a search tree will reduce the accesses to points P^B. By including the idea of EB, we can early-stop the search process to reduce redundant computation. Since this technique is only viable for the HD algorithm, directly invoking a standard k-NN search library will miss opportunity of applying this optimization, but *X-HD* seizes the opportunity to reduce unnecessary computation.

3.2.2 Utilizing HD Estimators. Nutanong et al. proposed an early HD algorithm that uses the branch-and-bound search principle to avoid visiting some tree branches in the index [57]. Unfortunately, their method is inherently sequential and thus cannot be used in massive parallel environments. However, the utilization of Hausdorff distance estimators in their paper motivated us to further prune redundant computation.

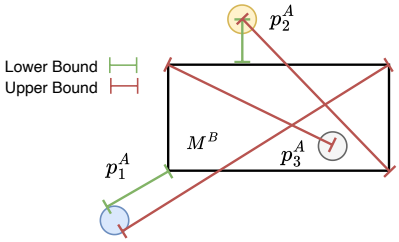


Figure 3: Using HD estimators to calculate the lower and upper bounds of point-MBR distance. p_3^A is within the MBR, so the lower bound is zero

The HD estimators provide the lower bound ($MinDist(p^A, M^B)$) and the upper bound ($MaxDist(p^A, M^B)$) of the distance between a given point p^A and the points bounded by a Minimum Bound Rectangle (MBR) M^B [57]. As illustrated in Fig. 3, *MinDist* is the distance from p^A to the closest face of M^B, while *MaxDist* is the distance to the farthest corner. The estimators allow us to eliminate redundant computations. We can cull all the points within M^B if they cannot contribute to the HD. For example, if the upper bound of the HD is not greater than *cmax*, we can safely skip all the points enclosed by M^B because they cannot increase *cmax*. We will elaborate on how to use the HD bounds to reduce computation workloads in §4.

3.3 Avoiding Heavy Workload in Shaders

OptiX adopts a single-ray programming model, where a single thread casts a ray and executes `IsIntersection` shader in serial. A thread will take a long execution time if it involves a large workload, therefore, underutilizing the GPU resources. Unfortunately, OptiX does not allow the use of warp/block synchronization intrinsics, which makes it impossible to use various load-balancing techniques, such as Cooperative Thread Array (CTA) scheduling [45, 51, 59]

Given a uniform grid, due to the inherent nature of real-world datasets, the points are often not evenly distributed across the grid cells. When handling the points within a cell, a single thread will serially visit them to calculate distances. While this may not be a problem for cells containing very few points, it can be disastrous for cells with tens of thousands of points, leading to a long running time. Due to the limitations of the RT programming model, the only way to address this issue is to offload the distance computation in RT shaders to general CUDA kernels, which allows us to use various synchronization primitives to rebalance the workloads.

4 The X-HD Algorithm

4.1 Algorithm Overview

X-HD first uses a uniform grid as the first-level index to partition points P^B, and spatially proximal points will be grouped into their cells. The second-level index, a BVH tree, is built over non-empty cells to accelerate the search for cells potentially containing the nearest neighbor. Rays are cast from P^A to find the cells hit by these rays (Fig. 4, Stage 1). Each `IsIntersection` shader invocation indicates the ray is close to a cell that potentially contains the nearest neighbor. Within the shader, we estimate the lower and upper bounds of the point-MBR distance, which helps prune

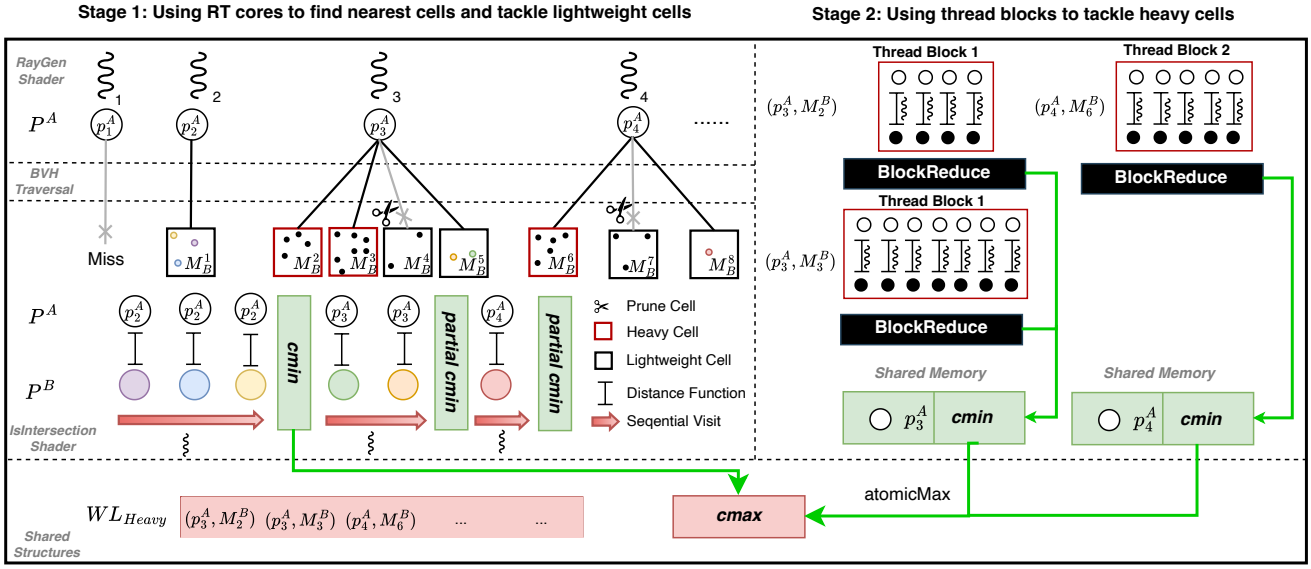


Figure 4: Execution process of *X-HD*. Stage 1: A grid is built over P^B to partition spatially proximal points into cells. Rays are cast from P^A search ray-AABB intersections with RT cores. `IsIntersection` shader sequentially processes each point in the cell to maintain the nearest neighbor. Stage 2: heavy cells are staged to an Intersection Queue, and they are processed with a dedicated CUDA kernel for better load balance.

non-contributing points, reducing redundant computation. Upon the execution of `IsIntersection`, we visit every point in the cell to pinpoint the nearest neighbor. Since some cells in real-world datasets may contain a large volume of points, sequentially visiting every point in such a cell is prohibitively slow. To mitigate the impact of imbalanced workloads, we do not immediately process these cells. Instead, we record the intersections in a worklist (WL_{Heavy}) and later process them in a dedicated CUDA kernel (Fig. 4, Stage 2), where thread blocks cooperatively process the cells for better load balance.

4.2 Iterative Nearest Neighbor Search

X-HD is an iterative algorithm that searches for the nearest neighbor by gradually growing the search radius r to reduce unnecessary intersections [56]. An iteration consists of building a BVH, casting rays, handling intersections, and adjusting the search radius, and Fig. 4 shows the first three steps. The iterations will repeat until every point $p^A \in P^A$ finds its nearest neighbor.

Build BVH. Initially, points P^B are inserted into a uniform grid G to place them into different cells, as shown in Fig. 2(a). We calculate the MBR for each non-empty cell, and these MBRs are then expanded by a search radius r, forming AABBs as inputs to build a BVH. Using the MBR instead of the cell boundary provides a tighter boundary, which helps to reduce false-positive intersections and more accurate HD bounds.

Cast Rays. Each point p^A casts a short ray to search for ray-AABB intersections, where the ray origin is p^A, and the direction can be arbitrary. Searching for intersections is implemented by traversing the BVH, which is accelerated by RT cores. This process is therefore very fast.

Handle Intersections. The `IsIntersection` shader is triggered when a ray hits an AABB. In the shader, we retrieve the AABB ID to locate its corresponding cell in the grid and calculate the distances between p^A and the points in the cell, where the minimum distance, *cmin*, is also maintained. After visiting all points in the cell, we use *cmin* to update the global maximum *cmax* with *atomicMax*. Some cells may not contribute to the HD, we prune them in this process.

Adjust Search Range. Some points in P^A may not hit any AABBs at all due to an insufficient search radius. Consequently, we increase r for a wider search range. Fig. 2(b) visualizes this process with a cell containing three points from P^B and two query points, p_1^A and p_2^A, from P^A. In *Iteration 1*, an AABB is created by extending the MBR by an initial search radius r_0, but it does not cover either p_1^A or p_2^A. The search radius is then increased to $r_0 + r_d$ to include more points from P^A, where r_d is how long the search radius grows in each iteration. In *Iteration 2*, p_2^A intersects the AABB, but p_1^A does not. The radius is further increased to $r_0 + 2 * r_d$. Finally, in *Iteration 3*, p_1^A intersects the AABB, and the search stops.

Algorithm 2 summarizes the above three steps. Line 1-5 is for building the BVH. Two worklists, WL_{In} and WL_{Miss}, are used to maintain the points casting rays and the points that cannot find intersections when the current iteration ends, respectively. In Line 11, rays are cast from the points in WL_{In} to find intersections, which is accelerated by RT cores. Subsequently, `IsIntersection` shaders will be invoked whenever an intersection is found, in which the distances between p^A and points in the cell will be calculated (Algorithm 3). If the ray from a point does not hit any AABB, the point will be moved to WL_{Miss}, which will be searched again with an expanded search radius in the next iteration. Cells with too many points will not be immediately processed in `IsIntersection`

shader. They are instead addressed with a dedicated kernel for better load balance (Algorithm 5). Therefore, we cannot use *cmin* to update *cmax* because the computation is incomplete. We stage the incomplete *cmin* values to *partial_cmin*, allowing the CUDA kernel `TackleHeavyCells` to resume the computation (§4.4). In Line 24, we increase the radius for a larger search area, which is explained in §5. In Line 26-27, we update the BVH with a new radius using the BVH refit function, which is much faster than rebuilding the BVH [28]. The above process repeats until WL_{In} is empty, and *cmax* will be the result of HD. Since the algorithm is a variant of NN search, its complexity remains $O(M \cdot N)$. However, the adoption of various pruning techniques makes *X-HD* performant in practice.

Algorithm 2: *X-HD* Outline

```
   Input  : Point Set A, Point Set B
   Output : Hausdorff Distance
1  Insert points P^B into a uniform grid G
2  MBRs ← GetNonEmptyCells(G)
3  r_0 ← GetInitRadius()
4  AABBs ← {M^B | (M^B_min − r_0, M^B_max + r_0), M^B ∈ MBRs}
5  BVH ← BuildBVH(AABBs)
6  WL_In = P^A, WL_Miss ← ∅, cmax ← 0
7  while WL_In ≠ ∅ do
8      partial_cmin ← ∅, WL_Heavy ← ∅
9      for each p^A in WL_IN do
10         cmin ← ∞
11         optixTrace(p^A, cmin)          // Cast a ray at p^A; cmin is a
             reference to be updated in IsIntersection
12         Handle Intersections in IsIntersection shader  // automatically
             invoked by RT cores (see Algorithm 3)
13         if the ray does not hit any cell then
14             WL_Miss ← WL_Miss ∪ p^A
15         else if the ray does not hit any heavy cell then
16             if cmin ≠ ∞ then
17                 atomicMax(&cmax, cmin) // Update only if the ray is
                     not terminated
18             end
19         else
20             partial_cmin ← partial_cmin ∪ cmin
21         end
22     end for
23     cmax ← TackleHeavyCells(partial_cmin, cmax, WL_Heavy)
24     r ← GetNextRadius(r, WL_In, WL_Miss)     // New search radius
25     WL_In ← WL_Miss, WL_Miss ← ∅
26     AABBs ← {M^B | (M^B_min − r, M^B_max + r), M^B ∈ MBRs}
27     BVH ← RefitBVH(AABBs)  // Update the BVH with new AABBs
28 end
29 return cmax
```

4.3 Culling Non-contributing Points

Not every point will contribute to the final Hausdorff distance. Given a point $p^A \in P^A$ and a subset of points from P^B that an MBR M^B tightly encloses, we prune either p^A or all the points in M^B if they cannot increase the value of *cmax* by considering these cases:

- Case 1: $MaxDist(p^A, M^B) \leq cmax$. If the upper bound of the distance from point p^A to any point in M^B is not greater than *cmax*, then these points cannot increase *cmax* and will be discarded.
- Case 2: $MinDist(p^A, M^B) \geq cmin$. If the lower bound of the distance from p^A to M^B is not less than the current minimum distance (*cmin*) of p^A, the points in the MBR cannot decrease the minimum distance further.

Algorithm 3: IsIntersection RT Shader (called by RT cores)

```
   Input  : Point p^A, Heavy cells WL_Heavy, global maximum cmax
   Output : Updated cmin, WL_Heavy
1  cmin ← getPayload // Get previous cmin from payload register
2  i ← GetPrimitiveIndex()        // index of the intersected AABB
3  M^B ← MBRs[i]
4  // WL_Heavy is updated by CalculateMinDist in Algorithm 4
5  min_dist = CalculateMinDist(p^A, M^B, cmax, cmin, WL_Heavy)
6  if min_dist = ∞ then
7      TerminateRay()          // Stop BVH traversal as p^A is culled
8  else
9      cmin = min(cmin, min_dist)
10     Write back cmin to the payload
11 end
```

- Case 3: $MinDist(p^A, M^B) > r$. If the lower bound of the distance is larger than the current search radius (r), the points in the MBR are outside the search area and can be pruned.

For *Case 1*, we should cull p^A because the distance from p^A to its nearest neighbor will not be greater than *cmax*; continuing the search for the nearest neighbor will not increase *cmax*. For *Case 2* and *Case 3*, we should cull all the points in M^B because they do not help to reduce *cmin*. If a point-MBR pair passes the above checks, we still have the opportunity to cull p^A again with EB.

Incorporating Early Break. The EB and the NN algorithms are not mutually exclusive. We further optimize *X-HD* by incorporating the EB strategy to prune points that will not affect the final HD value. For instance, if the distance from p^A to any point in M^B is no greater than *cmax*, p^A can be safely discarded.

Algorithm 4: CalculateMinDist

```
   Input  : Point p^A, MBR M_B, Current minimum distance cmin, Global
            maximum distance cmax, Search Radius r, Worklist WL_Heavy
   Output : Updated current min distance from p^A to points in M_B
1  if MaxDist(p^A, M^B) ≤ cmax then
2      return ∞                                   // Prune p^A by Case 1
3  else if MinDist(p^A, M^B) ≥ cmin then
4      return cmin                                // Prune M_B by Case 2
5  else if MinDist(p^A, M^B) > r then
6      return cmin                                // Prune M^B by Case 3
7  end
8  if |M^B| <offloading threshold then
9      for each p^B in M^B do
10         dist ← ||p^A, p^B||
11         if dist ≤ r then
12             cmin ← min(cmin, dist)
13         end
14         if dist ≤ cmax then
15             return ∞                           // Early break
16         end
17     end for
18 else
19     WL_heavy ← WL_heavy ∪ (p^A, M^B)            // Heavy cell
20 end
21 return cmin
```

We incorporate the three conditions into a distance computation function, as listed in Algorithm 4. In Line 1-7, the return of an infinity value means we should discard p^A, and the return of *cmin* without computation indicates the points in the MBR should be discarded. After passing these checks, it is worthwhile to visit the

points in the MBR to calculate distances. Line 8 checks whether the number of points in M^B is lower than a given threshold to determine whether we should visit the point in the shader. From Line 9 to Line 17, each point in the cell will be sequentially visited to calculate the minimum distance. Whenever we find the distance of a pair of points that is no greater than *cmax*, we can stop the computation and discard p^A according to the early break condition (Line 14-16). If a cell contains too many points, we postpone the computation and store the intersection (p_i^A, M_j^B) in the worklist WL_{Heavy} for later processing in a dedicated CUDA kernel.

4.4 Offloading Heavy Cells

Introducing a uniform grid also leads to an imbalanced workload issue. Considering that real-world datasets are not uniformly distributed, some cells may contain very few points ("Lightweight Cells" in Fig. 4), while some cells may contain a large volume of points ("Heavy Cells"). Due to the constraints of the RT cores' single-ray programming model, we have to sequentially process the points in the cell with the shader, which results in poor performance.

To improve load balance, we only offload the work of heavy cells to a dedicated CUDA kernel. One might wonder why only offload heavy cells instead of processing all intersections entirely with the CUDA kernel. There is a trade-off between using the RT shader and the CUDA kernel to calculate distances. If we entirely offload the distance computation to the kernel, we have no chance to update the global maximum *cmax* during the BVH traversal. Therefore, we cannot timely prune points with the early break optimization, leading to more redundant computation. In practice, we set the threshold to the load-balancing kernel's thread block size (calculated by `cudaOccupancyMaxPotentialBlockSize`) to strike a balance between a better load balance and high occupancy. Because offloading cells with fewer points than the block size would leave threads idle, it is more efficient to process these lightweight cells serially directly in the RT shader.

Algorithm 5 shows the CUDA kernel for processing heavy cells. The outermost for loop iterates over each point in WL_{Heavy} with a thread block. In Line 4, we retrieve the incomplete minimum distance *cmin* computed in `IsIntersection` shader. Both p^A and *cmin* are stored in shared memory for fast processing. After that, the entire thread block will cooperatively visit each intersected MBR M^B to maintain *cmin*. In Line 7, each thread in the block takes a point in M^B and calculates the distance. A block reduction primitive is used to aggregate the minimum distance collected from each thread. If any distance is no greater than *cmax*, we will notify the entire thread block to break the loop and give up the processing of all following intersections. In Line 22, we synchronize the block to ensure *cmin* is ready, and it is then applied to *cmax* with the *atomicMax* instruction.

Algorithm 5: TackleHeavyCells CUDA Kernel

```
   Input  : Incomplete minimum distance partial_cmin, Global maximum
            distance cmax, Worklist WL_Heavy
   Output : Updated cmax
1  for each thread block takes point p^A ∈ WL_Heavy do
2      __shared__ point_t p^A
3      __shared__ float cmin
4      cmin = partial_cmin[p^A] // Resume computation from the
           incomplete cmin computed by InIntersection shader
5      __syncthreads()
6      for each MBR M^B ∈ WL_Heavy that p^A intersects do
7          p^B ← M^B[threadIdx.x]    // Each thread takes a point p^B
8          dist ← ||p^A, p^B||
9          min_dist = BlockReduce(dist)
10         if threadIdx.x = 0 then
11             if min_dist ≤ cmax then
12                 cmin ← ∞                    // Flag down early break
13             else
14                 cmin ← min(cmin, min_dist)
15             end
16         end
17         __syncthreads()
18         if cmin = ∞ then
19             break                  // Thread block exits the loop
20         end
21     end for
22     __syncthreads()
23     if threadIdx.x = 0 and cmin ≠ ∞ then
24         atomicMax(&cmax, cmin)
25     end
26 end for
```

5 Implementation Details

Finding Initial Radius. We use the lower bound of $\overrightarrow{H}(P^A, P^B)$ as the starting search radius r_0 because this value represents the smallest possible value of search radius. The lower bound can be easily calculated using the MBRs of P^A and P^B [57].

Growing Search Radius. We grow the search radius by r_d after each iteration. If r_d is very small, many iterations may be needed to lift the radius to an appropriate value that allows p^A to hit an extended cell, and each iteration is not free. It has to cast rays and adjust the BVH for the new search radius. Conversely, if r_d is very large, we may only need one more iteration to complete the algorithm. However, we could face excessive intersections, which would be more costly than the overhead of adjusting the BVH.

We propose an adaptive radius growing strategy based on our observation. The size of the worklist WL_{In} decreases sharply at the beginning of the iterations. After a few iterations, the size of WL_{In} changes slowly. This can be explained by the presence of a few outliers that are very far from the dataset center. Based on this observation, we should adjust r according to the changing rate of WL_{In}. If the size of WL_{In} sharply decreases, we should also increase r quickly to reduce iterations. If WL_{In} decreases slowly, we should carefully increase r to avoid too many unnecessary intersections.

Algorithm 6 explains this idea. We set four predefined expansion factors: $\{8, 4, 2, 1\}$. If the worklist size decreases slowly, e.g., by less than $\frac{1}{8}$, r is increased quickly by eight times of the grid cell diagonal. Note that the search radius r must not exceed the upper bound of HD, so we constrain its value in Line 9.

Adaptive Grid Sizing. The grid resolution greatly impacts the performance of *X-HD*. If the grid is too sparse (low resolution), the number of cells is also low. Consequently, the number of intersections will be low, spending less time on BVH traversal. However, the number of points in cells will be much larger. In the extreme case of a grid having only a single cell, *X-HD* effectively acts as the EB solution. Conversely, if the grid resolution is very high, the number of intersections will also be very high, but each cell contains a minimum number of points, resulting in fewer nearest

Algorithm 6: GetNextRadius

Input : Current search radius r, Current worklist WL_{In}, Next iteration worklist WL_{Miss}
Output : New search radius r

$wl\_reduce\_ratio \leftarrow \frac{|WL_{In}|-|WL_{Miss}|}{WL_{In}}$
$radius\_expand\_factors \leftarrow \{8, 4, 2, 1\}$
for each $expand\_factor$ **in** $radius\_expand\_factors$ **do**
  if $wl\_reduce\_ratio < \frac{1}{expand\_factor}$ **then**
    $r \leftarrow r + expand\_factor * cell\_diagonal$
    break
  end
end for
$r \leftarrow min(r, MaxDist(M^A, M^B))$ // Max value of the radius
return r

neighbor candidates. Therefore, the best grid resolution depends on the trade-off between the cost of the BVH search and the cost of the distance calculation for candidate points in the cell. To approximate the optimal grid size, we define a cost function $C(l)$ for query point p^A based on the cell side length l, as shown in Equation 2.

$$C(l) = c_{hit} * I + c_{dist} * P \tag{2}$$

c_{hit} is a constant cost associated with a ray hitting an MBR, and c_{dist} is a constant cost of calculating the distance between a pair of points. I is the number of intersections of the ray originated at p^A, and P is the total number of points covered in the search area. Since we extend the MBRs by r, the actual volume indexed is $(l + 2r)^n$, where n is the number of dimensions, and the volume of a cell is l^n. A single query point will intersect multiple logical cells when $l \ll r$ because the extended MBRs will overlap. Therefore, I can be approximated with the ratio of their volumes (Equation 3).

$$I = \frac{(l + 2r)^n}{l^n} = (1 + \frac{2r}{l})^n \tag{3}$$

P can be approximated by the search volume multiplied by the point density of P^B, which is listed in Equation 4.

$$P = \rho(l + 2r)^n \tag{4}$$

Summing the above components, we have Equation 5:

$$C(l) = c_{hit} \cdot (1 + \frac{2r}{l})^n + c_{dist} \cdot \rho(1 + 2r)^n \tag{5}$$

To find the minimum cost $C(l_{opt})$, we take the derivative of $C(l)$ with respect to l and set it to zero $\frac{dC}{dl} = 0$, as $C(l)$ is strictly convex. This yields Equation 6:

$$l_{opt} = (\frac{2 \cdot r \cdot c_{hit}}{\rho \cdot c_{dist}})^{\frac{1}{n+1}} \tag{6}$$

Equation 6 intuitively describes that if the ratio (c_{hit}/c_{dist}) is high, we should increase l to reduce intersections. If the distance calculation is expensive (e.g., high-dimensional space), we should use a finer grid to narrow down candidate points. In practice, we use the initial radius r_0 as r, and c_{hit} and c_{dist} are determined using the clock() instruction with a profiling kernel when *X-HD* is launched, which makes it adaptive to different GPUs. We evaluate the effectiveness of this method in §6.6.

6 Evaluation

6.1 Settings

6.1.1 Baselines. We only consider exact HD solutions for a fair comparison, as *X-HD* is a general exact HD algorithm.

ITK. HD has a very important application in medical imaging to evaluate the performance of a segmenting algorithm [53, 67, 69]. We include the state-of-the-art industrial medical imaging segmentation tool, the Insight Toolkit (*ITK*), as a baseline. *ITK* supports parallel processing on the CPU. Note that the algorithm that *ITK* used is exclusively designed for processing voxels [48]. It cannot process floating numbers or two datasets with different sizes, so we only run MRI datasets on it.

NN-KD. *NN-KD* is a CUDA-based nearest neighbor search implementation using a KD-tree, and we use the *cukd* library as it is a high-quality implementation [72]. The introduction of this baseline is to show that a tree-based method without pruning techniques on the GPU is not very efficient.

NN-Clover. *Clover* is the state-of-the-art KNN library for GPU. *Clover* adopts a spatial-graph-based method, and it claims that it is generally faster than the RT-based method [32]. We evaluate its performance for the HD algorithm. The pruning technique also cannot be used in *Clover*, as it is designed for general KNN search.

EB. *EB* represents a GPU-based early break implementation of the HD algorithm. *EB* is very GPU-friendly due to its simple logic flow. There is no widely recognized *EB* solution available, so we implement it by ourselves.

RT-HDIST. *RT-HDIST* is the latest and the only work that exploits RT cores to compute HD [37]. *RT-HDIST* differs from us in the following ways.

(1) Different Grid Design. *RT-HDIST* adopts a quantized grid to group points. The quantized grid maps all floating-point coordinates to an integral range of $[0, 2^k]$, where k is the bit count that determines the grid size. Subsequently, a cluster of points within a cell is represented by the cell's top-left corner, which is called a "representative point." By using the grid, *RT-HDIST* creates AABBs centered only at the representative points instead of at every individual data point.
(2) No pruning techniques. Since all coordinates are converted into integers, the original boundary information of the cells is lost. Therefore, their quantized grid design cannot benefit from various pruning techniques like in *X-HD*.
(3) No load balance considerations.
(4) No adaptive radius adjustment. *RT-HDIST* simply grows the radius by the length of the cell diagonal in each iteration.
(5) No automatic grid-sizing. *RT-HDIST* relies on the users to manually tune the grid size.

6.1.2 Parameters. For *NN-KD*, it has a parameter to decide the leaf size; we use its default value of 8. *Clover* has a compile-time parameter H, which is the number of hubs. We found that $H = 256$ yields overall good performance. *RT-HDIST* has a parameter called bit count to determine the resolution of its quantized grid. For graphics datasets, we use the parameters presented in their paper [37]. For the MRI datasets, we use its default setting, which is 8. *X-HD* does not require any hand-tuned parameters.

6.1.3 Datasets. We collected a diverse set of datasets, including MRI images, geospatial datasets, and graphics models. The MRI datasets are from the Multimodal Brain Tumor Segmentation Challenge 2020 (BraTS) [5, 6, 50], comprising 494 images. Each image includes different types of tissue relaxation time. The locations of non-empty voxels are extracted as the coordinates. The names of datasets starting with *US* are US-based maps, such as the boundaries of counties [12], zip codes [17], census blocks [18], and water areas [16]. *OSMLakes* and *OSMParks* represent the boundaries of lakes and parks worldwide [15]. These large-scale spatial datasets are widely used for performance evaluation of spatial data processing systems [3, 15, 27, 73]. The graphics models come from The Stanford 3D Scanning Repository [13, 25, 39, 70]. We include them for an equivalent evaluation as they are also used in *RT-HDIST*. We also include the statistics of the datasets, such as the number of points. To reflect the skewness of the datasets, we use a uniform grid to partition the datasets and calculate the Gini index of the number of points in the cells, a commonly adopted method to describe the skewness of data distribution [49]. From the statistics, we can see that MRI datasets have a very low skewness (low Gini index), while the graphics and geospatial datasets can be very skewed.

6.1.4 Hardware. We use an NVIDIA RTX 3090 for the GPU-based artifacts. *ITK* is the only CPU-based software, and we evaluate it on a workstation equipped with an Intel i9-12900.

6.1.5 Measuring. For GPU-based methods, we start timing when the datasets are loaded onto the GPU and stop timing when the HD is produced. Note that no profilers are currently available to profile the utilization of RT cores [47, 55]. Therefore, we use the total running time and the number of intersections to analyze the execution patterns for RT-based methods.

6.2 Overall Performance

Fig. 5(a) shows the running time distribution for the BraTS datasets. Since BraTS has hundreds of images, we follow an evaluation method similar to Taha et al. [67] by randomly selecting 250 pairs of images and plotting the running time distributions. For example, a data point at $x = 20$ and $y = 5$ indicates that the running time for no more than 20% of the data points is no more than 5ms. We first examine the industrial baseline, *ITK*, which exhibits a very flat running time line, meaning it is less susceptible to data distributions, with an average and median running time of 46.8ms and 46.5ms, respectively. *NN-KD* produced performance numbers very similar to *ITK*, despite being a GPU-based method. *NN-Clover* and *RT-HDIST* are the least performant methods for these datasets. Since *EB* is a GPU-based brute-force solution with pruning techniques and is very efficient on the GPU, it is the best baseline in the majority of cases. However, the running time after the 95^{th} percentile is very long because, in these cases (similar distances between points), the EB condition is rarely triggered. Finally, *X-HD* consistently outperforms all baselines for all data points due to our efficient utilization of RT cores, pruning techniques, and better load balancing. The average running time is 8.9ms, and the median time is 8.4ms, showing that our method is less susceptible to data distributions, a similar trend to *ITK*. We will analyze why *X-HD* is so performant in §6.3 and §6.4.

Table 1: Datasets and their statistics. The Gini index is used to describe the skewness of the data.

Dataset	Category	#Dims	#Points or (max,avg,median)	Gini Index or (max,avg,median)
BraTS	494 MRI images	3D	(1.9M,1.5M,1.5M)	(0.17, 0.14, 0.14)
USCounty	Geospatial	2D	9.4M	0.77
USZipcode	Geospatial	2D	43.9M	0.61
Lakes	Geospatial	2D	301.7M	0.76
Parks	Geospatial	2D	403.7M	0.76
USWater	Geospatial	2D	22.8M	0.61
USCensus	Geospatial	2D	52.3M	0.65
All Nodes	Geospatial	2D	2.7B (Partially used)	-
Dragon	Graphics	3D	0.4M	0.42
AsianDragon	Graphics	3D	3.6M	0.38
HappyBuddha	Graphics	3D	0.5M	0.46
ThaiStatuette	Graphics	3D	4.9M	0.47

Fig. 5(b) shows the performance numbers on the geospatial datasets, which have a large number of points and high skewness. We could not include *RT-HDIST* because it cannot finish the experiment within an hour. We first look at *NN-KD*, which took 1176ms on *USWater-USBlock* datasets, with 94.8% of the time spent on building the KD-tree. *Clover*'s NN search took about 3150ms, with 63ms to search for the closest hub and 2.3s to search for nearest points in the hub. The GPU-based *EB* took about 535ms, which is about 2.2× faster than *NN-KD*. Finally, *X-HD* only took 83ms, achieving 6.4× speedups over *EB* due to our comprehensive optimization techniques. Fig. 5(c) shows the performance numbers over medium-sized and skewed datasets. We can see that *X-HD* is the fastest except in one case, *Dragon-Asian Dragon*, where *EB* took about 15.4ms while *X-HD* took 16.2ms. For the *Dragon-Buddha*, *Thai-Asian Dragon*, and *Thai-Buddha* datasets, *X-HD* outperforms *EB* with speedups of 1.2×, and 3.9×, 1.3×, respectively. The performance of *RT-HDIST* is not competitive due to excessive ray-AABB intersections and distance computations, even with RT cores. As a result, *X-HD* outperforms *RT-HDIST* with speedups of 10.5×, 38.3×, 17.9×, and 90.2×, respectively, for these four pairs of datasets.

6.3 Effectiveness of Pruning Techniques

We measure the running time as we gradually enable the optimizations, as shown in Fig. 6(a). To better analyze the behaviors of the optimizations, we measure the number of MBRs being hit (*Intersections*) and the number of pairs of points used in the distance calculation functions (*Visited Point Pairs*), as listed in Fig. 6(b). **No-Opt** stands for a vanilla uniform grid without any optimizations, which took about 8 seconds on *Dragon-Asian Dragon* datasets, where the number of intersections is about 165 million and 1.8×10^{11} pairs of points were involved in the distance calculation. Note that *No-Opt* is not equivalent to *RT-HDIST* due to the different grid design. After enabling **EB**, the running time reduced to about 2 seconds, the number of intersections went down to 21 million, and the number of point pairs was reduced to 2.3×10^{10}. We can see that by enabling EB, both the intersections and visited points are reduced because some points from P^A are discarded as they cannot increase *cmax*. After enabling the **Prune** optimization with the help of HD estimators, the number of intersections almost stayed the same. However, the number of points visited was reduced by more than two orders of magnitude, to about 589 million. With HD estimators, we can skip many points that cannot reduce

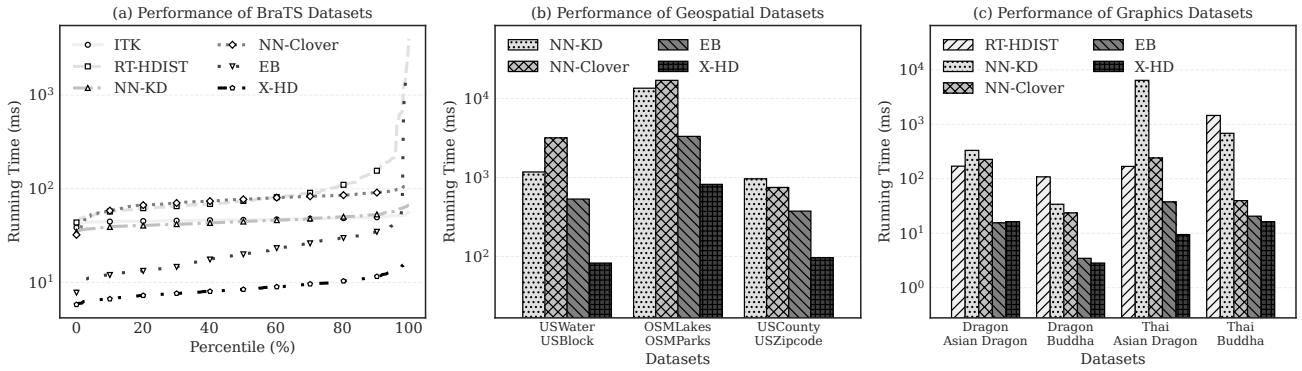


Figure 5: Comparing Overall Performance of Hausdorff Distance Solutions. (a) Running time distribution of six method on BraTS datasets; (b-c) End-to-end execution time on geospatial and graphics datasets

cmin. Since *RT-HDIST* does not have any pruning techniques, it results in more intersections than our method, which is very costly to process. However, the number of points visited by *RT-HDIST* is lower than ours, which could be a finer grid they used. Searching for an intersection is more expensive than processing a pair of points. For example, we empirically measured the former step using the `clock()` API, which takes about 1300 cycles on an RTX 3090, while the latter takes only about 260 cycles. As a result, *X-HD* outperforms *RT-HDIST* by 3.1× (without enabling load balance).

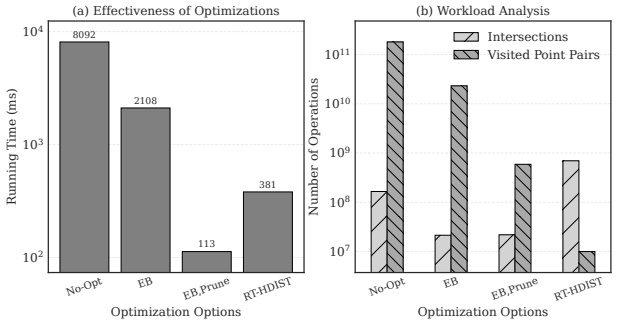


Figure 6: (a) Running time analysis by enabling optimization options on *Dragon-Asian Dragon* datasets; (b) Breakdown workloads by counting the number of intersections and the number of points involved in distance computation

6.4 Effectiveness of Load-balance Techniques

We analyze the effectiveness of the load balance techniques with geospatial and graphics datasets because they result in very skewed grids. Fig. 7(a) shows the breakdown of the total execution time with the option of load-balancing optimization. *OSMLakes-OSMParks* is the dataset most susceptible to the load-imbalance issue due to the skewness of the grid. Without enabling the offloading kernel to improve load balance, the RT shader took 3390ms to visit points in the grid cells and calculate point distances. After enabling the offloading kernel, the RT shader only took 223ms and the additional CUDA kernel took 254ms. The total time for the RT shader and the kernel is only 477ms, which is 7.1× faster than using the RT shader alone. Fig. 7(b) shows that offloading heavy cells is always beneficial, producing speedups of 7.6×, 2.9×, 4.4×, and 2.5×.

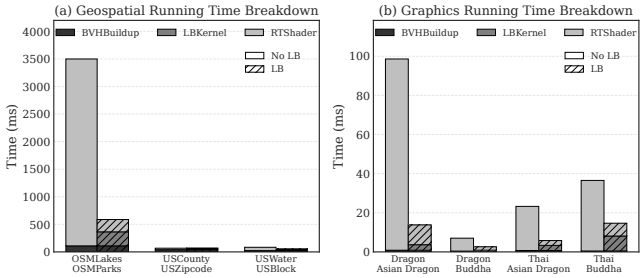


Figure 7: (a) Running time breakdown of *X-HD* and comparing the effectiveness of using load-balancing (LB) on geospatial datasets; (b) The same experimental settings but with graphics datasets.

6.5 Evaluation of Radius Growing Strategies

Fig. 8 compares three radius growing methods. "Add by Diagonal" method increases the search radius by the length of the cell diagonal for every iteration, which is used by *RT-HDIST* [37]. "Double Radius" method doubles the search radius, which is used by *RT-KNNS Unbound* [56]. We can see that "Double Radius" favors the geospatial datasets, while "Add Diagonal" shows consistent high-performance on graphics datasets. By looking into the running logs of the two methods, we found that "Add Diagonal" took 300 iterations on the *OSMLakes-OSMParks* datasets due to the conservative radius growing strategy, while "Double Radius" only took 7 iterations. Since the extent of geospatial datasets tends to be very high, a slowly growing radius will result in many iterations to converge. In turn, the aggressive growing strategy of "Double Radius" causes too many unnecessary intersections, resulting in consistently poor performance on the graphics datasets. Finally, our method adaptively grows the radius according to the changing ratio of the worklist, making it consistently good for all datasets.

6.6 Evaluation of Adaptive Grid Sizing

To evaluate the effectiveness of the grid-tuning technique in §5, we compare the running time to a grid with an optimal grid size for

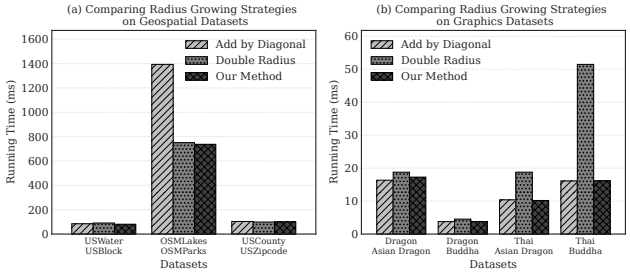


Figure 8: Comparing three radius growing strategies. "Add Diagonal" increases the radius by the length of the cell diagonal; "Double Radius" doubles the search radius

each dataset. The optimal grid size is determined with a brute force search, which is prohibitively expensive and cannot be accepted in practice. The two curves in Fig. 9(a) show the running times for BraTS datasets. We can see that the running time from an auto-sizing grid is only slightly slower than the optimal grid resolution, with an average and median slowdown of 12.9% and 5.1%, respectively. For the worst case, the auto-sizing grid is 123% slower than the optimal grid resolution. The very close performance numbers indicate that our cost-based grid-tuning method is highly accurate.

Fig. 9(b) and (c) compare the running times between the auto-sizing grid and the optimal grid on geospatial and graphics datasets. For *USWater-USBlock*, the optimal grid took 74ms while the grid with adaptive sizing took 83ms, which is 12.2% slower than the optimal grid. For the largest *OSMLakes-OSMParks* dataset, the optimal grid took 665ms, while using our method took 821ms, experiencing a slowdown of 23.5%. For the graphics datasets, the slowdowns of the auto-sized grid are 35.7%, 4.1%, 25.9%, and 15.7%, respectively.

6.7 Scalability

We evaluate how these methods behave as we increase input size. We used a subset of the *All Nodes* dataset [15]. P^A and P^B are from the same dataset, where P^B is translated along the x-axis by 0.5% to make them not completely the same. Fig. 10(a) shows the runtime of *EB-GPU*, *NN-RT*, and *X-HD* as the number of points increases. When the number of points is 12.5M, *NN-KD* and *NN-Clover* took 7444ms and 450ms, respectively, and *EB* took 283ms. *X-HD* only took 68ms, which 4.2× faster than the best baseline. Subsequently, we doubled the input size, and the running time of *NN-KD* increased 4.3×. *EB* and *NN-Clover* methods exhibit more favorable scalability profiles, increasing by about 1.9×. While *X-HD* increased by a factor of 2.6×, it is still significantly faster than all other methods. When increasing the input size from 100M to 200M, the running time of *NN-KD* increased about 2.3×. *EB* increased 2.5×. While *NN-Clover* increased 2.1×. *X-HD* increased by about 2.9×, which falls within the range of its $O(n \log n)$ time complexity.

6.8 Sensitivity to Overlap

The performance of HD algorithms can be susceptible to data distribution. For example, the NN-based algorithms favor two highly overlapping datasets, while the EB-based algorithm favors two non-overlapping datasets [67, 79]. We aim to evaluate how *X-HD* behaves by changing the degree of overlap between two datasets. Fig. 10(b) reports the running times as we gradually change the degree of overlap. This change is achieved by translating P^B along the x-axis by a distance proportional to 0.01% up to 0.6% of the extent of P^B's MBR. When the two datasets are highly overlapping (0.01%), *EB* performs the worst because the EB condition is rarely satisfied, deteriorating into a brute-force solution with a running time of 7561ms. In such a case, the NN-based method is favorable due to the less intensive index search. As the degree of overlap between the two datasets decreases, the running time of *EB* gradually decreases. Conversely, the running time of *NN-KD* increases because more branches in the KD-tree must be searched. *NN-Clover* is less susceptible to the degree of overlap, as its running time remains almost stable. *X-HD*, being an NN-based method incorporated into EB, is also less susceptible to the degree of overlap as it incorporates the advantages of these two algorithms.

6.9 Memory Footprint

Maintaining a low memory footprint is crucial for processing large datasets within the limited device memory. The EB algorithm requires no temporary memory, as it relies on the double-loop execution pattern. Conversely, NN-KD, NN-Clover, and *X-HD* require auxiliary memory to construct spatial indices. Among these, NN-KD generally consumes the least memory because data points can be naturally structured into a compact KD-tree.

Fig. 11 illustrates the temporary memory footprints across the geospatial and graphics datasets. As shown in Fig. 11(a), *X-HD* requires slightly more memory than NN-KD on the larger geospatial datasets. For example, on the OSMLakes-OSMParks dataset, NN-KD occupies 3080 MB of memory, while *X-HD* uses 3929 MB. For *X-HD*, this allocation breaks down into 9.5 MB for the BVH, 1542.3 MB for the grid, and 2301.8 MB for the worklist used to track heavy cells. Averaging the memory usage across the geospatial datasets, NN-KD consumes 1271 MB, NN-Clover requires 3614 MB, and *X-HD* uses 1549 MB—which is only marginally higher than the KD-tree.

On the graphics datasets, *X-HD* exhibits a more dynamic memory footprint. Because the BVH and grid require significantly less space for these inputs, most of the memory is allocated to the worklists, which may vary substantially. On average, NN-KD consumes 23.8 MB, NN-Clover takes 60.5 MB, and *X-HD* uses 46.0 MB. Given the substantial performance improvements offered by *X-HD*, this slight memory overhead is well justified.

7 Related Work

Hausdorff Distance. Nutanong et al. first applied branch-and-bound to accelerate HD computation [57], introducing a best-first search with a hierarchical priority queue. Taha et al. proposed the EB technique, achieving near-linear average complexity on medical images [67]. Zhang et al. [79] classified datasets into three categories and designed a hybrid framework to select algorithms accordingly. Chen et al. addressed EB inefficiency with a local-start search [10], while Ryu et al. improved pruning with points-ruling-out and random sampling [65]. These methods primarily target CPUs and rely on structures unsuitable for GPUs. Distorch [64] provides a GPU library, but it estimates the 95% HD, mainly for

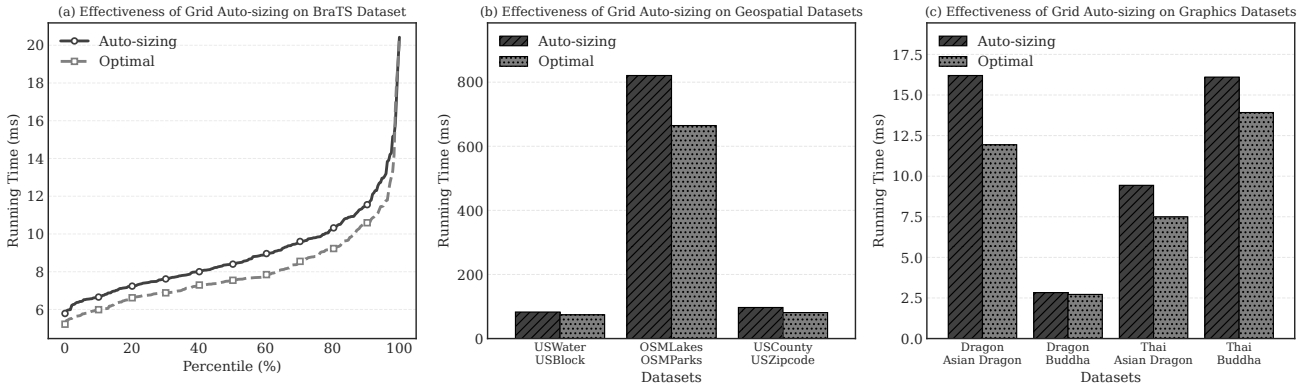


Figure 9: Comparing the running time of a grid with the adaptive sizing technique to the optimal grid on all datasets.

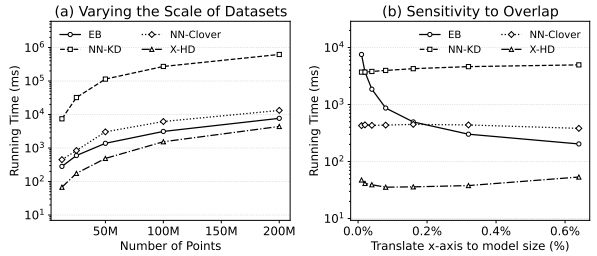


Figure 10: (a) Evaluation of scalability by increasing points; (b) Sensitivity to overlap by separating two datasets

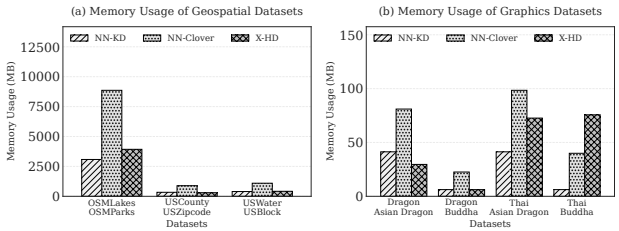


Figure 11: (a) Memory usage of geospatial datasets; (b) Memory usage of graphics datasets

medical applications. MedVoxelHD is a GPU-accelerated morphological Hausdorff, which only supports voxel processing and does not support Euclidean distance [53].

Distance Computation on the GPU gDist is a GPU-based parallel algorithm that efficiently computes the minimum/maximum Euclidean distance between general triangle meshes through improved traversal and culling techniques [22]. Their problem is related but distinct from ours: HD calculates the maximum of minimum distances for each point, while gDist calculates the minimum of minimum distances and the maximum of maximum distances.

Spatial Data Processing with RT Cores. Kim et al. compute point-to-face HD with RT cores [36], whereas we address point-set HD, a more general and computationally intensive problem. Evangelou et al. reformulated radius search as ray tracing [20], extended by RTNN for kNN with optimized scheduling [83], TrueKNN for arbitrary-radius kNN [56], and Arkade for non-Euclidean kNN [47]. RayJoin focuses on point-in-polygon and line segment queries [26]. LibRTS offers a general-purpose RT–accelerated spatial index with dynamic geometry updates [27].

Applications of Hausdorff Distance. HD has been widely applied in trajectories [29, 40, 41, 44, 62, 76–78, 80, 81], images [4, 11, 21, 31, 35, 54, 68, 82], geometry [2, 30, 38], and spatio-temporal data [43]. However, HD is usually treated as just one distance function, without dedicated optimization, leaving its high computational cost a bottleneck. Our approach makes HD practically viable for large-scale analysis by reducing this overhead.

Hausdorff Distance in Databases. PostGIS [52, 63] provides `ST_HausdorffDistance`, based on GEOS, but it is single-threaded and brute-force, lacking optimizations such as EB or NN and hardware acceleration. This results in poor performance on large or high-resolution datasets. Our algorithm addresses these limitations through algorithmic and hardware-based optimizations.

8 Conclusion

A central challenge in performance optimization for increasingly complex data analysis applications in the era of domain-specific architectures is how to effectively select, implement, orchestrate, and optimize algorithms to meet computational demands across heterogeneous hardware platforms. This paper introduces *X-HD*, the fastest Hausdorff Distance computation algorithm, which fully leverages both general-purpose and ray tracing cores in modern GPUs. The strength of *X-HD* lies in using hardware-acceleration and applying domain-specific optimizations at the same time, making it a unified solution capable of diverse workloads while maintaining consistently high performance. We believe that *X-HD* not only delivers a fast Hausdorff Distance computation tool for spatial database systems and other applications, but also serves as a compelling example of effective algorithm engineering for emerging heterogeneous hardware resources.

Acknowledgments

We would like to thank the anonymous reviewers for their constructive comments and suggestions. The work is supported in part by the U.S. National Science Foundation under grants CCF-2005884, CCF-2210753, CCF2312507, and OAC-2310510. Liang Geng is supported by a 2026 University Presidential Graduate Fellowship.

References

[1] Benjamin S Allen, James Anchell, Victor Anisimov, Thomas Applencourt, Abhishek Bagusetty, Ramesh Balakrishnan, Riccardo Balin, Solomon Bekele, Colleen Bertoni, Cyrus Blackworth, et al. 2025. Aurora: Architecting Argonne's First Exascale Supercomputer for Accelerated Scientific Discovery. *arXiv preprint arXiv:2509.08207* (2025).

[2] Nicolas Aspert, Diego Santa-Cruz, and Touradj Ebrahimi. 2002. Mesh: Measuring errors between surfaces using the hausdorff distance. In *Proceedings. IEEE international conference on multimedia and expo*, Vol. 1. IEEE, 705–708. doi:10.1109/ICME.2002.1035879

[3] Samuel Audet, Cecilia Albertsson, Masana Murase, and Akihiro Asahara. 2013. Robust and efficient polygon overlay on parallel stream processors. In *Proceedings of the 21st ACM SIGSPATIAL international conference on advances in geographic information systems*. 304–313.

[4] Kolawole O Babalola, Brian Patenaude, Paul Aljabar, Julia Schnabel, David Kennedy, William Crum, Stephen Smith, Tim F Cootes, Mark Jenkinson, and Daniel Rueckert. 2008. Comparison and evaluation of segmentation techniques for subcortical structures in brain MRI. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2008: 11th International Conference, New York, NY, USA, September 6-10, 2008, Proceedings, Part I 11*. Springer, 409–416. doi:10.1007/978-3-540-85988-8_49

[5] Spyridon Bakas, Hamed Akbari, Aristeidis Sotiras, Michel Bilello, Martin Rozycki, Justin S Kirby, John B Freymann, Keyvan Farahani, and Christos Davatzikos. 2017. Advancing the cancer genome atlas glioma MRI collections with expert segmentation labels and radiomic features. *Scientific data* 4, 1 (2017), 1–13. doi:10.1038/sdata.2017.117

[6] Spyridon Bakas, Mauricio Reyes, Andras Jakab, Stefan Bauer, Markus Rempfler, Alessandro Crimi, Russell Takeshi Shinohara, Christoph Berger, Sung Min Ha, Martin Rozycki, Marcel Prastawa, Esther Alberts, Jana Lipkova, John Freymann, Justin Kirby, Michel Bilello, Hassan Fathallah-Shaykh, Roland Wiest, Jan Kirschke, Benedikt Wiestler, Rivka Colen, Aikaterini Kotrotsou, Pamela Lamontagne, Daniel Marcus, Mikhail Milchenko, Arash Nazeri, Marc-Andre Weber, Abhishek Mahajan, Ujjwal Baid, Elizabeth Gerstner, Dongjin Kwon, Gagan Acharya, Manu Agarwal, Mahbubul Alam, Alberto Albiol, Antonio Albiol, Francisco J. Albiol, Varghese Alex, Nigel Allinson, Pedro H. A. Amorim, Abhijit Amrutkar, Ganesh Anand, Simon Andermatt, Tal Arbel, Pablo Arbelaez, Aaron Avery, Muneeza Azmat, Pranjal B., W Bai, Subhashis Banerjee, Bill Barth, Thomas Batchelder, Kayhan Batmanghelich, Enzo Battistella, Andrew Beers, Mikhail Belyaev, Martin Bendszus, Eze Benson, Jose Bernal, Halandur Nagaraja Bharath, George Biros, Sotirios Bisdas, James Brown, Mariano Cabezas, Shilei Cao, Jorge M. Cardoso, Eric N Carver, Adrià Casamitjana, Laura Silvana Castillo, Marcel Catà, Philippe Cattin, Albert Cerigues, Vinicius S. Chagas, Siddhartha Chandra, Yi-Ju Chang, Shiyu Chang, Ken Chang, Joseph Chazalon, Shengcong Chen, Wei Chen, Jefferson W Chen, Zhaolin Chen, Kun Cheng, Ahana Roy Choudhury, Roger Chylla, Albert Clérigues, Steven Colleman, Ramiro German Rodriguez Colmeiro, Marc Combalia, Anthony Costa, Xiaomeng Cui, Zhenzhen Dai, Lutao Dai, Laura Alexandra Daza, Eric Deutsch, Changxing Ding, Chao Dong, Shidu Dong, Wojciech Dudzik, Zach Eaton-Rosen, Gary Egan, Guilherme Escudero, Théo Estienne, Richard Everson, Jonathan Fabrizio, Yong Fan, Longwei Fang, Xue Feng, Enzo Ferrante, Lucas Fidon, Martin Fischer, Andrew P. French, Naomi Fridman, Huan Fu, David Fuentes, Yaozong Gao, Evan Gates, David Gering, Amir Gholami, Willi Gierke, Ben Glocker, Mingming Gong, Sandra González-Villá, T. Grosges, Yuanfang Guan, Sheng Guo, Sudeep Gupta, Woo-Sup Han, Il Song Han, Konstantin Harmuth, Huiguang He, Aura Hernández-Sabaté, Evelyn Herrmann, Naveen Himthani, Winston Hsu, Cheyu Hsu, Xiaojun Hu, Xiaobin Hu, Yan Hu, Yifan Hu, Rui Hua, Teng-Yi Huang, Weilin Huang, Sabine Van Huffel, Quan Huo, Vivek HV, Khan M. Iftekharuddin, Fabian Isensee, Mobarakol Islam, Aaron S. Jackson, Sachin R. Jambawalikar, Andrew Jesson, Weijian Jian, Peter Jin, V Jeya Maria Jose, Alain Jungo, B Kainz, Konstantinos Kamnitsas, Po-Yu Kao, Ayush Karnawat, Thomas Kellermeier, Adel Kermi, Kurt Keutzer, Mohamed Tarek Khadir, Mahendra Khened, Philipp Kickingereder, Geena Kim, Nik King, Haley Knapp, Urspeter Knecht, Lisa Kohli, Deren Kong, Xiangmao Kong, Simon Koppers, Avinash Kori, Ganapathy Krishnamurthi, Egor Krivov, Piyush Kumar, Kaisar Kushibar, Dmitrii Lachinov, Tryphon Lambrou, Joon Lee, Chengen Lee, Yuehchou Lee, M Lee, Szidonia Lefkovits, Laszlo Lefkovits, James Levitt, Tengfei Li, Hongwei Li, Wenqi Li, Hongyang Li, Xiaochuan Li, Yuexiang Li, Heng Li, Zhenye Li, Xiaoyu Li, Zeju Li, XiaoGang Li, Wenqi Li, Zheng-Shen Lin, Fengming Lin, Pietro Lio, Chang Liu, Boqiang Liu, Xiang Liu, Mingyuan Liu, Ju Liu, Luyan Liu, Xavier Llado, Marc Moreno Lopez, Pablo Ribalta Lorenzo, Zhentai Lu, Lin Luo, Zhigang Luo, Jun Ma, Kai Ma, Thomas Mackie, Anant Madabushi, Issam Mahmoudi, Klaus H. Maier-Hein, Pradipta Maji, CP Mammen, Andreas Mang, B. S. Manjunath, Michal Marcinkiewicz, S McDonagh, Stephen McKenna, Richard McKinley, Miriam Mehl, Sachin Mehta, Raghav Mehta, Raphael Meier, Christoph Meinel, Dorit Merhof, Craig Meyer, Robert Miller, Sushmita Mitra, Aliasgar Moiyadi, David Molina-Garcia, Miguel A. B. Monteiro, Grzegorz Mrukwa, Andriy Myronenko, Jakub Nalepa, Thuyen Ngo, Dong Nie, Holly Ning, Chen Niu, Nicholas K Nuechterlein, Eric Oermann, Arlindo Oliveira, Diego D. C. Oliveira, Arnau Oliver, Alexander F. I. Osman, Yu-Nian Ou, Sebastien Ourselin, Nikos Paragios, Moo Sung Park, Brad Paschke, J. Gregory Pauloski, Kamlesh Pawar, Nick Pawlowski, Linmin Pei, Suting Peng, Silvio M. Pereira, Julian Perez-Beteta, Victor M. Perez-Garcia, Simon Pezold, Bao Pham, Ashish Phophalia, Gemma Piella, G. N. Pillai, Marie Piraud, Maxim Pisov, Anmol Popli, Michael P. Pound, Reza Pourreza, Prateek Prasanna, Vesna Prkovska, Tony P. Pridmore, Santi Puch, Élodie Puybareau, Buyue Qian, Xu Qiao, Martin Rajchl, Swapnil Rane, Michael Rebsamen, Hongliang Ren, Xuhua Ren, Karthik Revanuru, Mina Rezaei, Oliver Rippel, Luis Carlos Rivera, Charlotte Robert, Bruce Rosen, Daniel Rueckert, Mohammed Safwan, Mostafa Salem, Joaquim Salvi, Irina Sanchez, Irina Sánchez, Heitor M. Santos, Emmett Sartor, Dawid Schellingerhout, Klaudius Scheufele, Matthew R. Scott, Artur A. Scussel, Sara Sedlar, Juan Pablo Serrano-Rubio, N. Jon Shah, Nameetha Shah, Mazhar Shaikh, B. Uma Shankar, Zeina Shboul, Haipeng Shen, Dinggang Shen, Linlin Shen, Haocheng Shen, Varun Shenoy, Feng Shi, Hyung Eun Shin, Hai Shu, Diana Sima, M Sinclair, Orjan Smedby, James M. Snyder, Mohammadreza Soltaninejad, Guidong Song, Mehul Soni, Jean Stawiaski, Shashank Subramanian, Li Sun, Roger Sun, Jiawei Sun, Kay Sun, Yu Sun, Guoxia Sun, Shuang Sun, Yannick R Suter, Laszlo Szilagyi, Sanjay Talbar, Dacheng Tao, Dacheng Tao, Zhongzhao Teng, Siddhesh Thakur, Meenakshi H Thakur, Sameer Tharakan, Pallavi Tiwari, Guillaume Tochon, Tuan Tran, Yuhsiang M. Tsai, Kuan-Lun Tseng, Tran Anh Tuan, Vadim Turlapov, Nicholas Tustison, Maria Vakalopoulou, Sergi Valverde, Rami Vanguri, Evgeny Vasiliev, Jonathan Ventura, Luis Vera, Tom Vercauteren, C. A. Verrastro, Lasitha Vidyaratne, Veronica Vilaplana, Ajeet Vivekanandan, Guotai Wang, Qian Wang, Chiatse J. Wang, Weichung Wang, Duo Wang, Ruixuan Wang, Yuanyuan Wang, Chunliang Wang, Guotai Wang, Ning Wen, Xin Wen, Leon Weninger, Wolfgang Wick, Shaocheng Wu, Qiang Wu, Yihong Wu, Yong Xia, Yanwu Xu, Xiaowen Xu, Peiyuan Xu, Tsai-Ling Yang, Xiaoping Yang, Hao-Yu Yang, Junlin Yang, Haojin Yang, Guang Yang, Hongdou Yao, Xujiong Ye, Changchang Yin, Brett Young-Moxon, Jinhua Yu, Xiangyu Yue, Songtao Zhang, Angela Zhang, Kun Zhang, Xuejie Zhang, Lichi Zhang, Xiaoyue Zhang, Yazhuo Zhang, Lei Zhang, Jianguo Zhang, Xiang Zhang, Tianhao Zhang, Sicheng Zhao, Yu Zhao, Xiaomei Zhao, Liang Zhao, Yefeng Zheng, Liming Zhong, Chenhong Zhou, Xiaobing Zhou, Fan Zhou, Hongtu Zhu, Jin Zhu, Ying Zhuge, Weiwei Zong, Jayashree Kalpathy-Cramer, Keyvan Farahani, Christos Davatzikos, Koen van Leemput, and Bjoern Menze. 2019. Identifying the best machine learning algorithms for brain tumor segmentation, progression assessment, and overall survival prediction in the BRATS challenge. *arXiv preprint arXiv:1811.02629* (2019). doi:10.48550/arXiv.1811.02629

[7] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jurgen Gall. 2019. Semantickitti: A dataset for semantic scene understanding of lidar sequences. In *Proceedings of the IEEE/CVF international conference on computer vision*. 9297–9307. doi:10.1109/ICCV.2019.00939

[8] John Burgess. 2020. Rtx on—the nvidia turing gpu. *IEEE Micro* 40, 2 (2020), 36–44. doi:10.1109/MM.2020.2971677

[9] Antonio Cardone, Satyandra K Gupta, Abhijit Deshmukh, and Mukul Karnik. 2006. Machining feature-based similarity assessment algorithms for prismatic machined parts. *Computer-Aided Design* 38, 9 (2006), 954–972. doi:10.1016/j.cad.2006.08.001

[10] Yilin Chen, Fazhi He, Yiqi Wu, and Neng Hou. 2017. A local start search algorithm to compute exact Hausdorff Distance for arbitrary point sets. *Pattern Recognition* 67 (2017), 139–148. doi:10.1016/j.patcog.2017.02.013

[11] Haili Chui and Anand Rangarajan. 2003. A new point matching algorithm for non-rigid registration. *Computer Vision and Image Understanding* 89, 2-3 (2003), 114–141. doi:10.1016/S1077-3142(03)00009-2

[12] Conservation.gov. 2025. US County Boundaries. https://hub.arcgis.com/datasets/17b89622df5643cda5339ae6649247a6_0. Feature Layer, ArcGIS Hub.

[13] Brian Curless and Marc Levoy. 1996. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 303–312. doi:10.1145/237170.237269

[14] Alex Dunn. 2019. *Tips and Tricks: Ray Tracing Best Practices*. NVIDIA Developer Blog. https://developer.nvidia.com/blog/rtx-best-practices/

[15] Ahmed Eldawy and Mohamed F Mokbel. 2015. Spatialhadoop: A mapreduce framework for spatial data. In *2015 IEEE 31st international conference on Data Engineering*. IEEE, 1352–1363. doi:10.1109/ICDE.2015.7113382

[16] Esri. 2025. USA Detailed Water Bodies. https://hub.arcgis.com/datasets/esri::usa-detailed-water-bodies/about. Feature Layer, ArcGIS Hub.

[17] Esri. 2025. USA ZIP Code Areas. https://hub.arcgis.com/maps/esri::usa-zip-code-areas/about. Web Map, ArcGIS Hub.

[18] Esri Federal Data. 2025. U.S. Census Blocks. https://hub.arcgis.com/datasets/d795eaa6ee7a40bdb2efeb2d001bf823_0/about. Feature Layer, ArcGIS Hub, utilizing data from the U.S. Census Bureau.

[19] Giannis Evagorou and Thomas Heinis. 2021. MAMBO-Indexing Dead Space to Accelerate Spatial Queries. In *Proceedings of the 33rd International Conference on Scientific and Statistical Database Management*. 73–84.

[20] Iordanis Evangelou, Georgios Papaioannou, Konstantinos Vardis, and Andreas A Vasilakis. 2021. Fast radius search exploiting ray-tracing frameworks. *Journal of Computer Graphics Techniques Vol* 10, 1 (2021), 25–48. http://jcgt.org/published/0010/01/02/

[21] Jianping Fan, D. K.Ỹ. Yau, A.K̃. Elmagarmid, and W.G̃. Aref. 2001. Automatic image segmentation by integrating color-edge extraction and seeded region growing. *IEEE Transactions on Image Processing* 10, 10 (Oct. 2001), 1454–1466. doi:10.1109/83.951532

[22] Peng Fan, Wei Wang, Ruofeng Tong, Hailong Li, and Min Tang. 2024. gDist: Efficient Distance Computation between 3D Meshes on GPU. In *SIGGRAPH Asia 2024 Conference Papers*. 1–11.

[23] William Randolph Franklin, Venkateshkumar Sivaswami, David Sun, Mohan Kankanhalli, and Chandrasekhar Narayanaswami. 1994. Calculating the area of overlaid polygons without constructing the overlay. *Cartography and Geographic Information Systems* 21, 2 (1994), 81–89.

[24] Yongsheng Gao and Maylor KH Leung. 2002. Face recognition using line edge map. *IEEE transactions on pattern analysis and machine intelligence* 24, 6 (2002), 764–779. doi:10.1109/TPAMI.2002.1008383

[25] Andrew Gardner, Chris Tchou, Tim Hawkins, and Paul Debevec. 2003. Linear light source reflectometry. *ACM Transactions on Graphics (TOG)* 22, 3 (2003), 749–758. doi:10.1145/882262.882342

[26] Liang Geng, Rubao Lee, and Xiaodong Zhang. 2024. RayJoin: Fast and Precise Spatial Join. In *Proceedings of the 38th ACM International Conference on Supercomputing*. 124–136. doi:10.1145/3650200.3656610

[27] Liang Geng, Rubao Lee, and Xiaodong Zhang. 2025. LibRTS: A Spatial Indexing Library by Ray Tracing. In *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*. 396–411. doi:10.1145/3710848.3710850

[28] Justus Henneberg and Felix Schuhknecht. 2023. RTIndeX: Exploiting Hardware-Accelerated GPU Raytracing for Database Indexing. *Proc. VLDB Endow.* 16, 13 (Sept. 2023), 4268–4281. doi:10.14778/3625054.3625063

[29] Dongxiao Hu, Lijun Chen, Hao Fang, Zhiwei Fang, Tao Li, and Yanjie Gao. 2024. Spatio-Temporal Trajectory Similarity Measures: A Comprehensive Survey and Quantitative Study. *IEEE Transactions on Knowledge and Data Engineering* 36, 5 (May 2024), 2191–2212. doi:10.1109/TKDE.2023.3323535

[30] Daniel P Huttenlocher, Klara Kedem, and Jon M Kleinberg. 1992. On dynamic Voronoi diagrams and the minimum Hausdorff distance for point sets under Euclidean motion in the plane. In *Proceedings of the eighth annual symposium on Computational geometry*. 110–119. doi:10.1145/142675.142700

[31] Daniel P Huttenlocher, Gregory A. Klanderman, and William J Rucklidge. 1993. Comparing images using the Hausdorff distance. *IEEE Transactions on pattern analysis and machine intelligence* 15, 9 (1993), 850–863. doi:10.1109/34.232073

[32] Victor Kamel, Hanxueyu Yan, and Sean Chester. 2025. CLOVER: A GPU-native, Spatio-graph-based Approach to Exact kNN. In *Proceedings of the 2025 International Conference on Supercomputing (ICS '25)*. ACM, Salt Lake City, UT, USA, 1–14. doi:10.1145/3721145.3730415

[33] Davood Karimi and Septimiu E Salcudean. 2019. Reducing the hausdorff distance in medical image segmentation with convolutional neural networks. *IEEE Transactions on medical imaging* 39, 2 (2019), 499–513.

[34] Amir Keramatian, Vincenzo Gulisano, Marina Papatriantafilou, and Philippas Tsigas. 2022. IP. LSH. DBSCAN: Integrated parallel density-based clustering through locality-sensitive hashing. In *European Conference on Parallel Processing*. Springer, 268–284.

[35] Hassan Khotanlou, Olivier Colliot, Jamal Atif, and Isabelle Bloch. 2009. 3D brain tumor segmentation in MRI using fuzzy classification, symmetry analysis and spatially constrained deformable models. *Fuzzy sets and systems* 160, 10 (2009), 1457–1473. doi:10.1016/j.fss.2008.11.016

[36] YoungWoo Kim, Sungmin Kwon, and Duksu Kim. 2025. RTPD: Penetration Depth calculation using Hardware accelerated Ray-Tracing. *arXiv preprint arXiv:2502.12463* (2025). doi:10.48550/arXiv.2502.12463

[37] Young Woo Kim, Jaehong Lee, and Duksu Kim. 2025. RT-HDIST: Ray-Tracing Core-based Hausdorff Distance Computation. In *Computer Graphics Forum*. Wiley Online Library, e70229.

[38] Vasileios Kopsachilis, Michalis Vaitis, Nikos Mamoulis, and Dimitris Kotzinos. 2020. Recommending Geo-semantically Related Classes for Link Discovery. *Journal on Data Semantics* 9 (2020), 151–177. doi:10.1007/s13740-020-00117-4

[39] Venkat Krishnamurthy and Marc Levoy. 1996. Fitting smooth surfaces to dense polygon meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 313–324. doi:10.1145/237170.237270

[40] Hai Lan, Jiong Xie, Zhifeng Bao, Feifei Li, Wei Tian, Fang Wang, Sheng Wang, and Ailin Zhang. 2022. VRE: a versatile, robust, and economical trajectory data system. *Proceedings of the VLDB Endowment* 15, 12 (Aug. 2022), 3398–3410. doi:10.14778/3554821.3554831

[41] Jae-Gil Lee, Jiawei Han, Xiaolei Li, and Hector Gonzalez. 2008. TraClass: trajectory classification using hierarchical region-based and trajectory-based clustering. *Proceedings of the VLDB Endowment* 1, 1 (Aug. 2008), 1081–1094. doi:10.14778/1453856.1453972

[42] Noppon Lertchuwongsa, Michele Gouiffes, and Bertrand Zavidovique. 2012. Enhancing a disparity map by color segmentation. *Integrated Computer-Aided Engineering* 19, 4 (2012), 381–397. doi:10.3233/ICA-2012-0413

[43] Ke Li, Lisi Chen, Shuo Shang, Haiyan Wang, Yang Liu, Panos Kalnis, and Bin Yao. 2022. Towards Controlling the Transmission of Diseases: Continuous Exposure Discovery over Massive-Scale Moving Objects. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI-22)*. International Joint Conferences on Artificial Intelligence Organization. doi:10.24963/ijcai.2022/540 Supported by NSFC (U21B2046, U2001212, 62032001, 61932004), Sichuan Science and Technology Program (2021YFS0007), and others.

[44] Ruiyuan Li, Huajun He, Rubin Wang, Sijie Ruan, Yuan Sui, Jie Bao, and Yu Zheng. 2020. TrajMesa: A Distributed NoSQL Storage Engine for Big Trajectory Data. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, Dallas, TX, USA, 2002–2005. doi:10.1109/ICDE48307.2020.00224

[45] Jiaxin Liu, Rubao Lee, Cathy H Xia, and Xia Odong Zhang. 2025. A Stable Marriage Requires a Shared Residence with Low Contention and Mutual Complementarity. In *2025 34th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 416–430.

[46] Helen Lockett and Marin Guenov. 2008. Similarity measures for mid-surface quality evaluation. *Computer-Aided Design* 40, 3 (2008), 368–380. doi:10.1016/j.cad.2007.11.008

[47] Durga Keerthi Mandarapu, Vani Nagarajan, Artem Pelenitsyn, and Milind Kulkarni. 2024. Arkade: k-Nearest Neighbor Search With Non-Euclidean Distances using GPU Ray Tracing. In *Proceedings of the 38th ACM International Conference on Supercomputing*. 14–25. doi:10.1145/3650200.3656601

[48] Calvin R Maurer, Rensheng Qi, and Vijay Raghavan. 2003. A linear time algorithm for computing exact Euclidean distance transforms of binary images in arbitrary dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25, 2 (2003), 265–270.

[49] Ke Meng, Jiajia Li, Guangming Tan, and Ninghui Sun. 2019. A pattern based algorithmic autotuner for graph processing on GPUs. In *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*. 201–213.

[50] Bjoern H Menze, Andras Jakab, Stefan Bauer, Jayashree Kalpathy-Cramer, Keyvan Farahani, Justin Kirby, Yuliya Burren, Nicole Porz, Johannes Slotboom, Roland Wiest, Levente Lanczi, Elizabeth Gerstner, Marc-André Weber, Tal Arbel, Brian B Avants, Nicholas Ayache, Patricia Buendia, D Louis Collins, Nicolas Cordier, Jason J Corso, Antonio Criminisi, Tilak Das, Hervé Delingette, Çağatay Demiralp, Christopher R Durst, Michel Dojat, Senan Doyle, Joana Festa, Florence Forbes, Ezequiel Geremia, Ben Glocker, Polina Golland, Xiaotao Guo, Andac Hamamci, Khan M Iftekharuddin, Raj Jena, Nigel M John, Ender Konukoglu, Danial Lashkari, José Antonió Mariz, Raphael Meier, Sérgio Pereira, Doina Precup, Stephen J Price, Tammy Riklin Raviv, Syed M S Reza, Michael Ryan, Duygu Sarikaya, Lawrence Schwartz, Hoo-Chang Shin, Jamie Shotton, Carlos A Silva, Nuno Sousa, Nagesh K Subbanna, Gabor Szekely, Thomas J Taylor, Owen M Thomas, Nicholas J Tustison, Gozde Unal, Flor Vasseur, Max Wintermark, Dong Hye Ye, Liang Zhao, Binsheng Zhao, Darko Zikic, Marcel Prastawa, Mauricio Reyes, and Koen Van Leemput. 2014. The multimodal brain tumor image segmentation benchmark (BRATS). *IEEE transactions on medical imaging* 34, 10 (2014), 1993–2024. doi:10.1109/TMI.2014.2377694

[51] Duane Merrill, Michael Garland, and Andrew Grimshaw. 2012. Scalable GPU graph traversal. *ACM Sigplan Notices* 47, 8 (2012), 117–128.

[52] Deng Min, Li Zhilin, and Chen Xiaoyong. 2007. Extended Hausdorff distance for spatial objects in GIS. *International Journal of Geographical Information Science* 21, 4 (2007), 459–475. doi:10.1080/13658810601073315

[53] Jakub Mitura, Beata E Chrapko, and Oliwia Bachanek-Mitura. 2024. MedVoxelHD: Improved CUDA-accelerated morphological Hausdorff distances in medical image analysis. *SoftwareX* 26 (2024), 101744.

[54] Frederic Morain-Nicolier, Stephane Lebonvallet, Etienne Baudrier, and Su Ruan. 2007. Hausdorff distance based 3D quantification of brain tumor evolution from MRI images. In *2007 29th Annual international conference of the IEEE engineering in medicine and biology society*. IEEE, 5597–5600. doi:10.1109/IEMBS.2007.4353615

[55] Vani Nagarajan, Rohan Gangaraju, Kirshanthan Sundararajah, Artem Pelenitsyn, and Milind Kulkarni. 2025. RT-BarnesHut: Accelerating Barnes-Hut Using Ray-Tracing Hardware. In *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*. 43–56. doi:10.1145/3710848.3710885

[56] 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*. 289–300. doi:10.48550/arXiv.2305.18356

[57] Sarana Nutanong, Edwin H Jacox, and Hanan Samet. 2011. An incremental Hausdorff distance calculation algorithm. *Proceedings of the VLDB Endowment* 4, 8 (2011), 506–517. doi:10.14778/2002974.2002978

[58] NVIDIA Corporation. 2024. NVIDIA OptiX 8.1 - Programming Guide. Available at: https://raytracing-docs.nvidia.com/optix8/guide/index.html. Accessed: 11 December 2025.

[59] Muhammad Osama, Serban D Porumbescu, and John D Owens. 2023. A programming model for GPU load balancing. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*. 79–91.

[60] Dimitris Papadias, Yufei Tao, Kyriakos Mouratidis, and Chun Kit Hui. 2005. Aggregate nearest neighbor queries in spatial databases. *ACM Transactions on Database Systems (TODS)* 30, 2 (2005), 529–576. doi:10.1145/1071610.1071616

[61] Shibin Parameswaran and Jeffrey Ellen. 2014. Identifying outliers in human movement trajectories clustered by hausdorff distance. In *Proceedings of the International Conference on Data Science (ICDATA)*. The Steering Committee of The World Congress in Computer Science, Computer …, 1.

[62] Jeff M. Phillips and Pingfan Tang. 2019. Simple Distances for Trajectories via Landmarks. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL '19)* (Chicago, IL, USA). Association for Computing Machinery, New York, NY, USA, 468–471. doi:10.1145/3347146.3359098

[63] PostGIS Development Team. 2025. ST_HausdorffDistance — PostGIS Documentation. https://postgis.net/docs/ST_HausdorffDistance.html Accessed: 2025-04-01.

[64] Jérôme Rony, Jonas Adler, James Lottes, Aapo Kyrola, Tri Huynh, and Adam Lerer. 2025. DisTorch: A fast GPU implementation of 3D Hausdorff Distance. In *Medical Imaging with Deep Learning (MIDL)*. https://openreview.net/forum?id=jv7DIZS4wG

[65] Jegoon Ryu and Sei-ichiro Kamata. 2021. An efficient computational algorithm for Hausdorff distance based on points-ruling-out and systematic random sampling. *Pattern Recognition* 114 (2021), 107857. doi:10.1016/j.patcog.2021.107857

[66] Dong-Gyu Sim, Oh-Kyu Kwon, and Rae-Hong Park. 1999. Object matching algorithms using robust Hausdorff distance measures. *IEEE Transactions on image processing* 8, 3 (1999), 425–429. doi:10.1109/83.748897

[67] Abdel Aziz Taha and Allan Hanbury. 2015. An efficient algorithm for calculating the exact Hausdorff distance. *IEEE transactions on pattern analysis and machine intelligence* 37, 11 (2015), 2153–2163. doi:10.1109/TPAMI.2015.2408351

[68] Abdel Aziz Taha and Allan Hanbury. 2015. Metrics for evaluating 3D medical image segmentation: analysis, selection, and tool. *BMC medical imaging* 15 (2015), 1–28. doi:10.1186/s12880-015-0068-x

[69] Kuo-Kun Tseng, Ran Zhang, Chien-Ming Chen, and Mohammad Mehedi Hassan. 2021. DNetUnet: a semi-supervised CNN of medical image segmentation for super-computing AI service. *The Journal of Supercomputing* 77, 4 (2021), 3594–3615.

[70] Greg Turk and Marc Levoy. 1994. Zippered polygon meshes from range images. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. 311–318. doi:10.1145/192161.192241

[71] Eleni Tzirita Zacharatou, Harish Doraiswamy, Anastasia Ailamaki, Claudio Silva, and Juliana Freire. 2017. GPU rasterization for real-time spatial aggregation over arbitrary polygons. *Proceedings of the VLDB Endowment* 11, 3 (2017). doi:10.14778/3157794.3157803

[72] Ingo Wald. 2022. GPU-friendly, parallel, and (almost-) in-place construction of left-balanced kd trees. *arXiv preprint arXiv:2211.00120* (2022).

[73] Congying Wang, Jia Yu, and Zhuoyue Zhao. 2023. GLIN: A (G) eneric (L) earned (In) dexing Mechanism for Complex Geometries. In *Proceedings of the 11th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*. 1–12.

[74] Yiqiu Wang, Yan Gu, and Julian Shun. 2020. Theoretically-efficient and practical parallel DBSCAN. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2555–2571.

[75] Yiqi Wu, Fazhi He, and Soonhung Han. 2017. Collaborative CAD synchronization based on a symmetric and consistent modeling procedure. *Symmetry* 9, 4 (2017), 59. doi:10.3390/sym9040059

[76] Dong Xie, Feifei Li, and Jeff M. Phillips. 2017. Distributed trajectory similarity search. *Proceedings of the VLDB Endowment* 10, 11 (Aug. 2017), 1478–1489. doi:10.14778/3137628.3137655

[77] Dingqi Yao, Gao Cong, Chengliang Zhang, and Jingjing Bi. 2019. Computing Trajectory Similarity in Linear Time: A Generic Seed-Guided Neural Metric Learning Approach. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, Macao, China, 1358–1369. doi:10.1109/ICDE.2019.00123

[78] Dongxiang Zhang, Zimu Chang, Shuangjie Wu, Yuan Yuan, Kian-Lee Tan, and Gang Chen. 2022. Continuous Trajectory Similarity Search for Online Outlier Detection. *IEEE Transactions on Knowledge and Data Engineering* 34, 10 (Oct. 2022), 4690–4704. doi:10.1109/TKDE.2020.3046670

[79] Dejun Zhang, Fazhi He, Soonhung Han, Lu Zou, Yiqi Wu, and Yilin Chen. 2017. An efficient approach to directly compute the exact Hausdorff distance for 3D point sets. *Integrated Computer-Aided Engineering* 24, 3 (2017), 261–277. doi:10.3233/ICA-170544

[80] Baihua Zheng, Lisi Weng, Xiaofeng Zhao, Kai Zeng, Xiaofang Zhou, and Christian S. Jensen. 2021. REPOSE: Distributed Top-k Trajectory Similarity Search with Local Reference Point Tries. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, Chania, Greece, 708–719. doi:10.1109/ICDE51399.2021.00067

[81] Silin Zhou, Shuo Shang, Lisi Chen, Christian S. Jensen, and Panos Kalnis. 2024. RED: Effective Trajectory Representation Learning with Comprehensive Information. *arXiv preprint arXiv:2411.15096* (Nov. 2024). doi:10.48550/arXiv.2411.15096 Accepted by VLDB 2025.

[82] Xingquan Zhu, Jianping Fan, Ahmed K. Elmagarmid, and Walid G. Aref. 2002. Hierarchical video summarization for medical data. In *Storage and Retrieval for Media Databases 2002 (Proceedings of SPIE, Vol. 4676)*, Minerva M. Yeung, Chung-Sheng Li, and Rainer W. Lienhart (Eds.). SPIE, San Jose, CA, USA. doi:10.1117/12.451110 Contact: {zhuxq,ake,aref}@cs.purdue.edu; jfan@uncc.edu.

[83] 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*. 76–89. doi:10.1145/3503221.3508409