Fast Radius Search Exploiting Ray-Tracing Frameworks

I. Evangelou G. Papaioannou K. Vardis A. A. Vasilakis
Dept. of Informatics, Athens University of Economics and Business, Greece

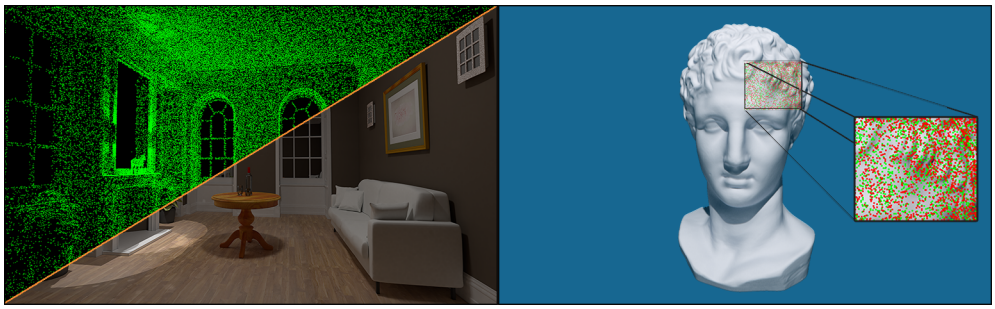

Figure 1. Several graphics applications that require intensive spatial search operations can be significantly accelerated by our method. Left: global illumination via progressive photon mapping. Photons are highlighted in green. Right: local point-cloud registration of partial surface scans shown as red and green points.

Abstract

Spatial queries to infer information from the neighborhood of a set of points are frequently performed in rendering and geometry-processing algorithms. Traditionally, these are accomplished using radius and k-nearest neighbors search operations, which utilize kd-trees and other specialized spatial data structures that fall short of delivering high performance. Recently, advances in ray-tracing performance, with respect to both acceleration data-structure construction and ray-traversal times, have resulted in a wide adoption of the ray-tracing paradigm for graphics-related tasks that spread beyond typical image synthesis. In this work, we propose an alternative formulation of the radius-search operation that maps the problem to the ray-tracing paradigm in order to take advantage of the available GPU-accelerated solutions for it. We demonstrate the performance gain relative to traditional spatial search methods, especially on dynamically updated sample sets, using two representative applications: geometry processing of point-wise operations on scanned point clouds and global illumination via progressive photon mapping.

 ISSN 2331-7418

1. Introduction

Many rendering and geometry-processing algorithms rely on spatial neighborhood queries on large point sets. Image-synthesis algorithms, like photon-mapping variants and many-light or radiance-caching methods are typical examples. In the context of geometry processing, point-cloud registration and local feature extraction are also prominent intensive tasks that operate on spatial neighborhoods or require the discovery of correspondence between nearest points. If performed frequently, all these operations can introduce a significant computational overhead, an issue that needs to be addressed in order to allow for fast performance and scalability of the intended application.

Most neighborhood queries utilize spatial hierarchies, such as kd-trees or regular structures (uniform or hash grids), to hierarchically subdivide sample points and accelerate query performance. Building high-quality data structures directly translates to higher query performance but generally impacts construction time negatively. Needless to say, from a development standpoint, the process of implementing an optimized data structure can become challenging.

Motivated by the importance of this problem, in this paper we demonstrate how to leverage a highly-optimized existing ray-tracing framework in order to efficiently map the radius-search task to ray traversal. Central to our approach is the idea of relating the query radius with samples and, as a result, treating them as regular primitives of known bounds instead of simple points. This allows us to store the point samples in an optimized data structure for ray tracing, available right off the shelf from the respective API and map the radius-search problem to the ray-tracing paradigm, relying on optimized and hardware-accelerated APIs to perform the queries fast and effortlessly. We demonstrate the significant performance boost and simplicity of such a generic approach through the implementation of an NVIDIA OptiX-based neighborhood-search mechanism and its application to various representative application scenarios, such as progressive photon mapping, point-cloud correspondence, and point-cloud normal estimation.

2. Related Work

Early literature reveals the need for efficient radius-search queries in image-synthesis tasks, such as irradiance caching [Ward et al. 1988], but perhaps the most representative family of image-synthesis methods that relies heavily on the efficiency of radius search, is photon mapping [Jensen 1996] and its variants. The need for exploring neighbor samples was also demonstrated in the context of bidirectional path-tracing methods [Georgiev et al. 2012] and virtual point lights [Sriwasansak et al. 2018]. More recently, nearest-neighbors searches have also been used to derive a selection probability for a candidate light during next event estimation [Vorba et al. 2019].

In the context of geometry processing and analysis, many local geometric descriptors, i.e., scalar or multi-dimensional quantities that characterize the geometric shape of a (sampled) neighborhood on a 3D surface or volume, require the extraction of either the k-nearest samples or the samples within a specific sphere radius that defines the scale of the operator. Since the family of actively used local descriptors is quite large, we refer the interested reader to the Point Cloud Library, where implementations of the most typical geometric descriptors and local feature-extraction methods on point clouds are provided [Rusu and Cousins 2011]. Another common task that heavily uses 1-nearest neighborhood search is local point-cloud registration, or the Iterative Closest Point algorithm [Besl and McKay 1992] and its derivative methods.

Below we briefly summarize the typical generic data structures used for the above tasks and present recent work on how ray tracing has been employed to assist in them.

2.1. Spatial Data Structures

The most prominent hierarchical data structure for nearest-neighbors searches is the kd-tree [Bentley 1975], which is extensively used in a variety of global-illumination tasks, especially those related to the photon-tracing scheme [Jensen 1996]. Kd-trees are also frequently employed in geometry-processing and recognition tasks, including the inference of local features [Hoppe et al. 1992] and point-cloud alignment [Zhang 1994]. Additionally, the performance of the hierarchy construction has been greatly improved by adapting it to a GPU architecture [Zhou et al. 2008] as well as its final tree quality based on the voxel volume heuristic [Wald et al. 2004]. Finally, the octree [Meagher 1982] and its many GPU implementations have also been exploited for nearest-neighbors searches in illumination tasks [Ward et al. 1988; Křivánek et al. 2008; Wang et al. 2019].

Attempts to optimize radius and nearest-neighbors searches were also demonstrated with grid-based data structures. Hash maps [Ma and McCool 2002] exploit a locality-sensitive hashing scheme to approximate nearest neighbors in a coherent and parallel manner. Despite the benefits, this approach suffers from efficiency issues when samples are not uniformly distributed, which is the typical case.

2.2. Hardware-accelerated Ray Tracing

Ray tracing on the GPU typically employs some specific type of a bounding volume hierarchy (BVH) as a ray-geometry acceleration data structure (ADS). With the increased popularity of ray tracing, several high-performance BVH data structures have been proposed that benefit from the GPU's parallel architecture [Lauterbach et al. 2009; Karras and Aila 2013; Domingues and Pedrini 2015]. In our case, this is particularly beneficial for many methods that utilize spatial neighborhood queries, since a BVH can have both a fast construction time and a competitive query performance, making it suitable for dynamic updates of the queried sample set. Even thought spatial

grids, like hash-grids, perform well on construction, the query performance can deteriorate in cases with uneven distribution of samples, which is rather common. On the other hand, while kd-trees solve this problem, they lack efficient construction algorithms for GPUs, whereas BVHs have been successful in balancing the performance of both tasks.

2.3. Applying Ray Tracing to Other Tasks

Recent enhancements in ray tracing, even at the hardware level, inspired researchers to creatively map problems to the ray-traversal paradigm, in order to exploit the fast performance and optimized emerging implementations.

First of all, treating point samples as primitives with volume in hierarchical data structures is not new. Fabianowksi et al. [2009] stored and queried photons, extended by an estimated splat radius, in a linear bounding-volume hierarchy [Lauterbach et al. 2009]. Here we generalize and abstract this idea further.

In the same spirit as in our work, Wald et al. [2019] exploit the OptiX framework to perform point-in-tetrahedron queries for volume visualization of unstructured shapes. Recently, Knoll et al. [2019] demonstrated how to exploit the OptiX API and hardware ray-tracing cores to render large particle data using marching rays that progressively accumulate intersected particles that are stored as generic primitives in a BVH. Concurrently and independently from our work, Zellmann [2020] proposed an efficient implementation of the spring-embedders algorithm, an iterative graph-drawing technique mapped to the ray-tracing paradigm. Similar to the previous methods, the OptiX framework is exploited for radius-search queries in the two-dimensional domain for all the vertices of a graph.

In this work we focus on solving spatial queries as a general-purpose task in a three-dimensional domain and show how this can be a competitive alternative to a regular GPU-accelerated kd-tree, both in terms of implementation convenience and overall performance.

3. Radius Search using Ray Tracing

3.1. Radius Search Formulation

A radius-search operation is defined by a set of points $S = \{\mathbf{s}_1, \mathbf{s}_2, \ldots\} \subseteq \mathbb{R}^3$ that represent the *sample space* and a set of points $Q = \{\mathbf{q}_1, \mathbf{q}_2, \ldots\} \subseteq \mathbb{R}^3$ that encompasses the *queries* to be performed. For every $\mathbf{q}_i \in Q$, the task is to find the subset of samples $S_{\mathbf{q}_i}$ that resides within a maximum search radius $r_i \in \mathbb{R}$, according to an indicator function:

$$I_{\mathbf{q}}(\mathbf{s}) = \begin{cases} 1, & d(\mathbf{s}, \mathbf{q}) \leq r, \\ 0, & \text{otherwise}, \end{cases} \tag{1}$$

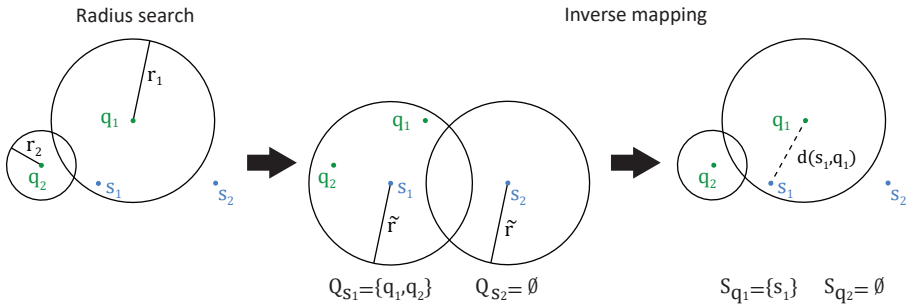

Figure 2. The forward and inverse radius-search process. The sample in query point-radius formulation (left) can be mapped to an inverse spatial query (middle), followed by a sample rejection step (right).

where $d(\mathbf{s}, \mathbf{q})$ is a distance function. Typically, this is the ℓ_2 norm. From Equation (1) and the symmetry property of $d(\mathbf{s}, \mathbf{q})$, the problem can be equivalently defined as locating for every $\mathbf{s}_j \in S$ all query points $Q_{\mathbf{s}_j} \subseteq Q$ according to the following indicator function:

$$I_{\mathbf{s}}(\mathbf{q}) = \begin{cases} 1, & d(\mathbf{s}, \mathbf{q}) \leq \tilde{r}, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

In simple terms, the query can be inverted by assigning a sufficiently large radius $\tilde{r} = \max_{\mathbf{q}_i \in Q}(r_i)$ for each sample $\mathbf{s}_j$. A subsequent *rejection* step is then performed to compute $S_{\mathbf{q}_i}$ from $Q_{\mathbf{s}_j}$ by ensuring that the original search radius is satisfied: $d(\mathbf{s}_j, \mathbf{q}_i) \leq r_i$, when $\mathbf{q}_i \in Q_{\mathbf{s}_j}$. A simple example of the inverse-mapping process is illustrated in Figure 2. Obviously, if the search radius is constant for all queries, we can set $\tilde{r} = r_i$, omitting the last check and greatly simplifying the query.

The above transformation of a gathering operation into an assignment one, has been also used in the case of photon mapping [Stürzlinger and Bastos 1997], where instead of determining the irradiance by gathering photons within a radius r_i around a camera ray-hit point $\mathbf{q}_i$, a photon-*splatting* operation assigns a photon $\mathbf{s}$ to hit points within a splat radius $\tilde{r}$.

In the same spirit as in the work of [Fabianowski and Dingliana 2009], here we exploit the inverse-search procedure in order to embed the set of samples S as points with radius $\tilde{r}$ in a fast spatial-acceleration structure, but we further take advantage of the hardware-accelerated ray-tracing mechanism to evaluate Equation (2). The particular problem re-formulation enables us to treat samples as primitives and, therefore, exploit existing BVH builders to store samples and accelerate queries (Sections 3.2 and 3.3) as well as explore different cost models that optimize them.

Typically, the final tree quality in terms of traversal performance is quantified using the surface area heuristic [MacDonald and Booth 1990] (SAH). On the other hand, a reasonable and often used criterion for spatial queries is the minimization of

the volume heuristic (VH), as we are interested in a point-in-volume probability measure according to our formulation in Section 3.1. In order for a BVH to be consistently efficient for spatial queries, we must guarantee that a structure that is optimized according to the SAH cost function also bounds the total volume heuristic cost. This turns out to be true, and an elementary proof can be found in the Appendix. Simply put, by minimizing the SAH, we always implicitly minimize the VH, meaning that by exploiting a fast-built SAH-based BVH, query performance does not fall short. Obviously, SAH is only an upper bound and therefore, a tree optimized with the VH can perform better, in theory.

Internal structure and indexing mechanics of trees built by modern BVH algorithms have some additional beneficial characteristics, aside from the parallel construction. First, the spatial coherence near the leaves offers infrequent tree-level changes on the lower tree levels during traversal. Second, since the input samples have relatively small bound extents, defects arising from node overlap during node splitting are less frequent, especially when the radius takes relatively small values and is progressively reduced, as is the case in progressive variants of photon mapping [Hachisuka et al. 2008]. Finally, an often incorrect assumption in the application of the SAH cost function, which requires that rays march unoccluded through the scene before hitting the target (internal) bounding volume, is actually true in our problem, which yields a more realistic and predictable performance.

In the following sections, given a set of query points $\mathbf{q}_i$, we explain how to identify either *all* samples, or only the k-nearest ones that reside within a query radius r_i, using an off-the-shelf ray-traversal framework.

3.2. Radius Search via Ray Traversal

Constructing and traversing the acceleration data structure is a two-step process. First, an axis-aligned bounding box (AABB) is constructed for every sample $\mathbf{s}_j$ based on $\tilde{r}$ and forwarded for a regular BVH tree construction. Second, for each query $\mathbf{q}_i$, a ray is defined with origin at $\mathbf{q}_i$ and an infinitesimal ray extent. For implementation purposes, we provide an arbitrary non-zero ray-direction vector $\vec{\mathbf{v}}$, since only the ray origin is required. A radius-search operation can then be directly mapped to an "intersection" ray-tracing event, and it is completed in a single ray-traversal invocation through ray-bounds intersection between the tree nodes and the ray. Since sample AABBs that are potentially within range of r_i must enclose it, by definition of our problem, the ray will eventually reach the leaves and correctly classify potential in-radius samples $\mathbf{s}_j$ according to $d(\mathbf{s}_j, \mathbf{q}_i)$. We summarize this method in Algorithm 1. Assuming object-based splits as well as sequential traversal, unique primitive records are guaranteed to be retrieved and, therefore, no additional modifications are required.

An alternative valid approach to the problem would be to store every query $\mathbf{q}_i$ and its associated search radii r_i as primitives in the BVH, instead. Rays constructed with

each one of the sample positions as origin would then be used to determine to which query points the samples contribute. However, this would require a BVH query for every sample, followed by an atomic update of the query's list of results, an overhead that is clearly avoided in the previous approach. GPU-accelerated BVH-traversal implementations are ray-parallel, with the traversal itself being sequential, which favors query-"gathering" operations, such as the one proposed here. On the other hand, parallel sample traversals attempting to concurrently update multiple query-result lists are inefficient.

Algorithm 1: Radius Search using BVH.

Input: Queries $Q = \{\mathbf{q}_i\}$, Samples $S = \{\mathbf{s}_j\}$.
Output: Search set $S_{\mathbf{q}_i} \subseteq S, \forall \mathbf{q}_i \in Q$.
begin
  $\tilde{r} \leftarrow \max_{\mathbf{q}_i \in Q}(r_i)$; $\vec{\mathbf{v}} \leftarrow (eps, eps, eps)$; $\{S_{\mathbf{q}_i} \leftarrow \emptyset\}$;
  $bvh \leftarrow$ BuildBVH($S, \tilde{r}$);
  forall $\mathbf{q}_i \in Q$ **do**
    $ray \leftarrow (\mathbf{q}_i, \vec{\mathbf{v}}, 0, eps)$; ▷ $(\mathbf{origin}, \mathbf{direction}, t_{min}, t_{max})$
    RayTrace(bvh, ray, $S_{\mathbf{q}_i}$, Intersect);
  end
  return $\{S_{\mathbf{q}_i}\}$;
end

Function `BuildBVH(`S, r`)`**:**
  $bvh \leftarrow$ EmptyBVH();
  forall $\mathbf{s} \in S$ **do**
    $aabb \leftarrow (\mathbf{s} - r \cdot \mathbf{1}^T, \mathbf{s} + r \cdot \mathbf{1}^T)$;
    bvh.AddElement($\mathbf{s}, aabb$);
  end
  return bvh;
end

Function `Intersect(`$\mathbf{s}$, ray, $S_{\mathbf{q}_i}$`)`**:**
  $(\mathbf{o}, \mathbf{d}, t_{min}, t_{max}) \leftarrow ray$;
  if $||\mathbf{s} - \mathbf{o}|| < r_i$ **then**
    $S_{\mathbf{q}_i} \leftarrow S_{\mathbf{q}_i} \cup \mathbf{s}$;
  end
end

3.3. Truncated k-nn Search via Ray Traversal

In the same spirit, we can exploit a BVH containing samples $\mathbf{s}_j$, each associated with a bound of radius $\tilde{r}$, for k-nearest-neighbors searches with a truncation distance $\tilde{r}$, beyond which all queried samples are rejected. Although this truncation comes out of necessity rather than choice, since $\tilde{r}$ affects the BVH construction and query performance and cannot be infinite, many practical algorithms already employ a truncation strategy to reject outliers and boost convergence rate. A typical example is the widely-used Iterative Closest Point alignment method, which frequently employs distance-based nearest-point culling for convergence speed and outlier rejection [Rusinkiewicz and Levoy 2001]. Since every sample is scanned over the predefined radius, as we already described in Section 3.2, an internal structure per query point of maximum size k can be maintained and we can track the k-nearest samples, effectively collecting the appropriate samples.

4. Evaluation

We assess the efficiency of the proposed radius-search method using the publicly available OptiX [Parker et al. 2010] ray-tracing API (version 7.2), on an NVIDIA GeForce RTX 2080 Ti graphics card with 11GB video memory and CUDA version 10.1.

```
#define FLT_EPSILON 1.e-16f

extern "C" __global__ void  __raygen__radSearch(void) {
    const uint3     index = optixGetLaunchIndex();
    const query_t & query = getQuery(index);
    const float3    dir   = make_float3(FLT_EPSILON, FLT_EPSILON,
                                                     FLT_EPSILON);
    const float     tmin  = 0.f;
    const float     tmax  = FLT_EPSILON;

    payload_t payload;
    payload.query = query;

    optixTrace(ADSHandle, query.pos, dir, tmin, tmax,
        0.f, OptixVisibilityMask(1),
        OPTIX_RAY_FLAG_DISABLE_CLOSESTHIT |
        OPTIX_RAY_FLAG_DISABLE_ANYHIT,
        0, 1, 0, payload);
}
```

Listing 1. OptiX program that invokes the radius-search process of each query.

```
extern "C" __global__ void __miss_radSearch(void)         {/*Empty*/}
extern "C" __global__ void __closesthit__radSearch(void) {/*Empty*/}
extern "C" __global__ void __anyhit__radSearch(void)      {/*Empty*/}

extern "C" __global__ void __intersection__radSearch(void) {
    query_t          & query      = payload.query;
    const uint32_t     primIndex  = optixGetPrimitiveIndex();
    const sample_t   & sample     = getSample(primIndex);
    const payload_t  & payload    = getPayload();
    const float3     & ray_origin = optixGetWorldRayOrigin();
    const float3       diff       = sample.pos - ray_origin;
    const float        t          = dot(diff, diff);

    if(t < payload.query.radius * payload.query.radius) {
#ifdef TRUNC_KNN // See Section 3.3
        if (t < payload.maxDistElem) {
            // Cache the current sample into k-closest array
            replaceFurthestElem(payload, query);

            // Find the element with the maximum distance
            // and record its index
            recordFurthestElem(payload, query);
        }
#else // Radius search, Section 3.2

    // Process sample within query radius...
#endif
    }
}
```

Listing 2. OptiX programs that issue query-sample intersections and record them.

As described in Section 3.2, we calculate the AABB for each sample, based on the input-radius parameter. These samples are then used to construct an OptiX acceleration data structure for which we enable compaction and fast-trace flags. Query points can then index it through the *ray-generation* program shown in Listing 1. In order to apply radius and truncated k-nn search as described in Section 3.2 and Section 3.3, only the *intersection* program is required. The latter will both validate and cache potential intersections between query and sample points effectively omitting any additional callable programs (see Listing 2). This is accomplished by disabling the closest and any-hit programs through the OptiX trace function during the ray-generation phase. The source code of our OptiX-based implementation can be found in the supplemental material 5. It should be noted that despite the API-specific flags used in our specific implementation, the process is generic enough to be applied to any other ray-tracing API, as long as custom primitive AABBs can be computed for the ADS construction stage and the intersection callable is a user-modifiable function.

We further evaluate our approach with the publicly available GPU-accelerated FLANN kd-tree [Muja and Lowe 2009], where we use the default hyper-parameters for each radius-search invocation, unless otherwise stated. In Section 4.1, we evaluate both frameworks in two distinct configurations of randomly generated point sets. Next, in Section 4.2, we evaluate the performance of our method in two common geometric operations that require k-nearest-neighbors searches. Finally, in Section 4.3 we validate the performance of our radius-search approach in progressive photon mapping [Hachisuka et al. 2008].

To level the ground for the comparison as much as possible, we perform the following steps: (1) In order to closely replicate the caching strategy of FLANN, we also use custom buffers storing per-query result sets for sample indices as well as distances for which we apply truncated k-nn as described in Section 3.3 and Listing 2; (2) we disable FLANN heap usage as well as the sorting operation of the resulting buffer.

4.1. Generic Radius-Search Evaluation

In this section, we measure the efficiency of radius search using the OptiX BVH and the FLANN kd-tree for storing and querying sample points under two generic configurations that build the corresponding hierarchy according to (1) a uniform distribution and (2) a Gaussian distribution with eight spherical modes. These two configurations serve as a qualitative baseline for both data structures. Typically, uniformly distributed samples will result in balanced trees which, in turn, greatly improve load balance and tree-search overhead of query invocations. On the other hand, trees generated according to the second configuration are generally less balanced. To assess the query performance of the constructed trees, we generate only uniformly distributed queries. The distribution of the query points is irrelevant to our tests; they are not stored in an acceleration structure and can be sorted and batched according to the specific requirements of an application, if further speedup is desired.

We evaluate and report the radius search performance for the above configurations in experiments of increasing radius values and two sets of sample populations, a *low-density* one, ranging from 10^4 to 10^5 and a *high-density* one, ranging from 10^5 to 10^6.

For sake of fairness, we pre-compute the maximum neighbor capacity needed among the queries prior to the execution of both frameworks in order to pre-allocate per-query index and distance matrices and, therefore, gather every potential sample.

Uniform sample distribution. Figure 3 summarizes the results for the radius search on uniformly distributed samples with low- and high-density sample populations, respectively. When the radius is relatively small, we can clearly observe that our approach outperforms FLANN in every setup between query and sample size (Figure 3 (top- and bottom-left)). In contrast, for larger radii, when samples and query size increase simultaneously (Figure 3 (top- and bottom-right)), the overall performance

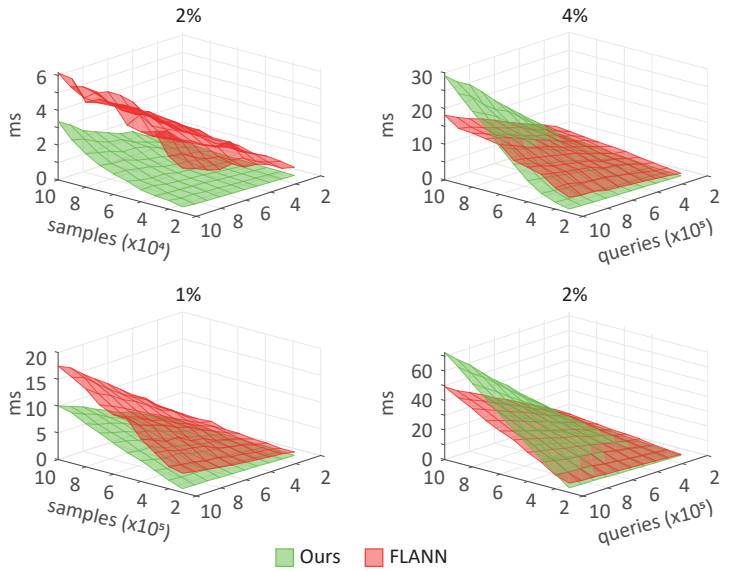

Figure 3. Radius search performance evaluation of uniformly distributed low-density samples (top row) and high-density samples (bottom row). The radius bandwidth, relative to the sample space bounding-box side, is indicated on top of each contour.

gain via the ray-tracing traversal is immediately negatively affected and as a result, the FLANN kd-tree becomes more efficient.

In Figure 4, we demonstrate the effect of the search radius for a relatively small fixed query (2^{15}) for both low- and high-density sample sets, in order for the gathering buffers to fit into GPU memory. In the first case (Figure 4 (left)), our method outperforms FLANN, despite the aggressive radius size. However, as the sample density increases, the performance deteriorates rapidly even for small radius values (Figure 4 (right)). Performance degradation is mainly caused by the increased overlap

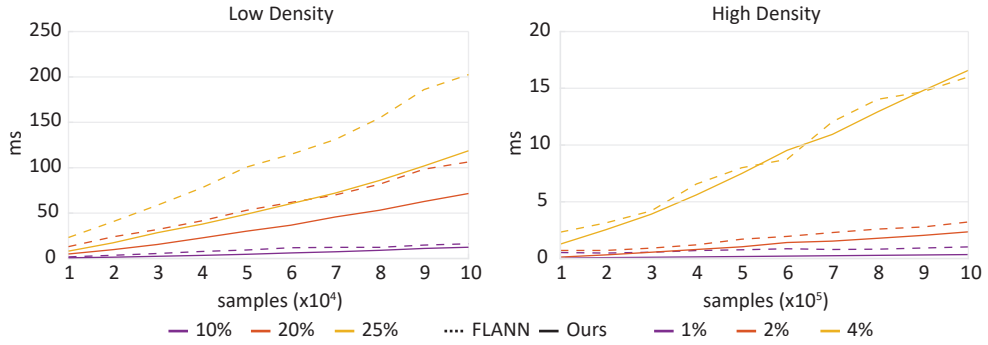

Figure 4. Radius of increasing size with constant query size $\left(2^{15}\right)$, for uniformly distributed samples of low density (left) and high density (right).

of sample AABBs as opposed to FLANN, which performs near optimal point-wise splits due to the spatial median split strategy it employs, effectively generating higher quality trees.

Gaussian sample distribution. In the second configuration, we measure the performance of simulating non-uniform distributions of samples; the results are presented in Figure 5. The multi-modal Gaussian distribution reflects a more realistic dispersion of samples in the queried space than the uniform case, since in practical-application scenarios, samples are either concentrated on the geometry surfaces (e.g., object point-clouds, particles) or form volumetric concentrations (e.g., particle systems and samples in participating media). As we can observe, our method performed better than FLANN in every combination of sample population and number of queries. Additionally, in contrast to the case of the uniform sample distribution, our method outperformed FLANN in all tests with varying radius, as shown in Figure 6 under similar query size configuration. In part, the significant performance gap is due to the different splitting strategies during the creation of the acceleration data structures as well as a relative overhead introduced, especially in the case of the FLANN kd-tree, from incoherent memory access during the result-set updates.

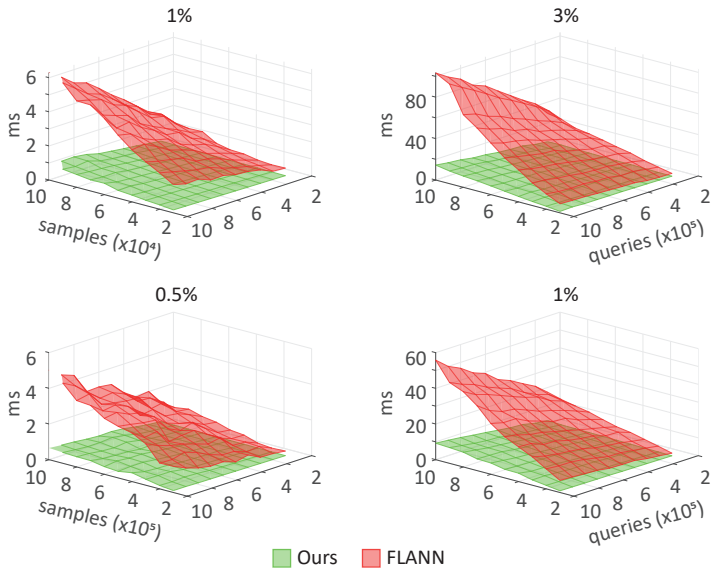

Figure 5. Radius-search evaluation for multi-modal Gaussian distributed samples with low density (top row) and high density (bottom row). The radius size, relative to the sample space bounding-box side, is depicted on top of each contour.

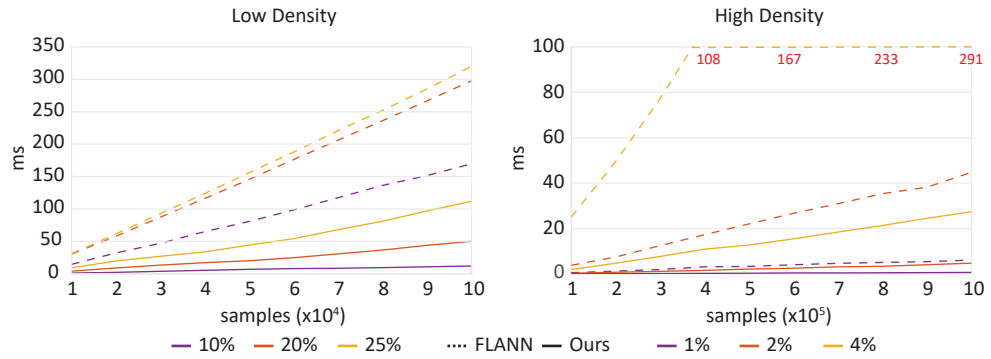

Figure 6. Radius-search timings as bandwidth progressively increases for multi-modal Gaussian samples with low density (left) and high density (right), with fixed size of queries $\left(2^{15}\right)$.

4.2. Geometry Processing Tasks

The task of point-cloud registration requires efficient nearest-neighbors searches. Given a set of points as a sample reference and a set of points as the query point-cloud, the objective is to evaluate for every point in the query subset, the closest corresponding point in the reference set and record the point-wise distance. The latter operation is often invoked via truncated k-nn search, based on a predefined radius and $k = 1$. In many modern applications, such as interactive digitization, the point clouds must be aligned and incrementally updated during the process, which demands both fast queries and short acceleration-structure build time.

We assess the performance of our method with the OptiX BVH builder compared to the FLANN kd-tree on this task in four distinct densely distributed point clouds that vary in size from 100K to 1M points. For each point cloud, we randomly subtract half the points and use them as the queries, while the remaining ones serve as the sample set from which we construct our tree hierarchy. This arrangement effectively models the scenario of nearest-point queries for partial surface-scan registration, where points in the dataset have similar populations and largely represent the same surface, albeit with different samples. The radius hyper-parameter is identical and pre-tuned for both frameworks to match the radius required to gather one sample on average. It should be noted that in these experiments, for both frameworks, we only allocate buffers corresponding to $k = 1$ for each query point, instead of trying to accommodate all samples in the query radius.

From the performance measurements presented in Figure 7 for $k = 1$, we can observe that our method's inference time is consistently faster in every case by at least $2.0\times$. Additionally, we can observe that, due to the superior tree-construction timings of OptiX BVH, as opposed to FLANN kd-tree, our method can produce interactive frame rates, when construction of the query point cloud is also necessary, in order to

	# points	$k = 1$	$k = 4$	$k = 8$
Build				
Lionhead	100K	1.4/21.4 (**14.9**×)	2.4/26.2 (**10.8**×)	
Metope	350K	3.3/29.1 (**8.6**×)	4.9/37.4 (**7.5**×)	
Dora block	500K	4.5/35.2 (**7.7**×)	6.5/42.4 (**6.5**×)	
Hermes	1M	7.8/45.4 (**5.8**×)	11.8/59.6 (**5.0**×)	
Search				
Lionhead	100K	0.1/0.4 (**3.3**×)	0.2/0.7 (**2.8**×)	0.5/0.9 (**1.7**×)
Metope	350K	0.2/0.7 (**2.8**×)	1.0/1.7 (**1.5**×)	1.9/2.6 (**1.3**×)
Dora block	500K	0.3/0.9 (**2.7**×)	1.6/2.6 (**1.5**×)	3.4/3.7 (**1.1**×)
Hermes	1M	1.1/2.2 (**2.0**×)	4.3/6.4 (**1.4**×)	8.1/8.2 (**1.0**×)

Figure 7. Build and Gather measurements comparing our OptiX-based radius-search method and FLANN on four point clouds tracking different number of nearest neighbors. The numbers in the parentheses indicate the performance improvement of our method over FLANN.

perform bi-directional point-wise closest point queries and when the query subset is progressively adapted (e.g., for live progressive surface scanning).

Another family of common geometric operations on point clouds is the estimation of local geometric features, such as curvature or surface normal orientation [Guennebaud and Gross 2007; Hoppe et al. 1992]. This task also typically requires local neighborhood determination on point clouds. For the simplest case of normal estimation, a reasonable approximation is often achieved when, for every point, the normal is estimated using the three-closest neighboring points. We evaluate the performance of the latter operation on the same set of point clouds described earlier. The radius hyper-parameter is tuned in a similar manner to the previous experiment. For this task, every point serves as a query and a sample at the same time and, therefore, we set the maximum number of neighbors required, k, equal to 4. We also repeat the experiment for $k = 8$ and set the radius for each point cloud accordingly.

Due to the small number of result records k per query ($k > 1$) allocated, buffer traversals and frequent updates are invoked from both frameworks during the search phase. However, as shown in Figure 7, for $k = 4$, we observe that our approach is still faster in every test. Setting $k = 8$, closed the gap in terms of relative inference time over FLANN. Nevertheless, data structure construction time for every point cloud remained significantly lower.

4.3. Progressive Photon Mapping

We further evaluated our radius-search method on progressive photon mapping by Hachisuka et al. [2008], extended with the global statistics formula proposed by Knaus et al. [2011]. This method invokes a bidirectional tracing scheme, from both the light sources and the virtual sensor, in order to approximate the energy equilibrium. A data structure, the *photon map*, is responsible for caching and indexing the particles that iteratively emanate in batches from the light sources.

Typically, due to multiple surface-scattering events, the total number of active photon records can vary from a few thousand to over a million in each frame. Additionally, in almost every case, the photon distribution after the end of the light-tracing phase will be highly non-uniform, due to convergent light paths and the termination of photons on scene-geometry surfaces. Consequently, several efficiency issues arise, making the gathering stage a non-trivial task to handle, since an additional data hierarchy is required in order to maintain these photons. Furthermore, progressive variants of the photon-mapping algorithm that process a new batch of photons in each iteration, require a reconstruction step of the whole hierarchy in every progressive step. This signifies the importance of efficiently performing *both* the initialization of the data structure and the gathering process, in the form of radius-search queries.

For our case study, we ran experiments mainly on indoor scenes and did not employ any path-length reduction strategy, such as Russian Roulette. As such, we maintained as many active photons as possible per frame and stressed the data-structure construction and access. Similar to the previous applications, we used the OptiX BVH to store the photon samples, which we queried with the camera-recorded hits and evaluated the performance of our gathering variant against the FLANN kd-tree on the same task.

For this set of experiments and for the OptiX approach only, we fully exploited the structure of this algorithm during the gathering phase and evaluated the photon density in-place during the intersection-program invocation (see Listing 2), effectively omitting the index and distance buffers, altogether. This greatly simplified the implementation and allowed the gathering of an arbitrary number of samples in the local neighborhood, as compared to the FLANN framework. This is also the reason why Gather times in the OptiX implementation are significantly smaller than the FLANN case. Please bear in mind that the same modifications could in theory be applied to the FLANN framework, but would require significant customization and algorithm insight to optimize, whereas in our case, they were implemented in a few lines of CUDA kernel code.

Since FLANN requires pre-allocated GPU buffer pointers for the radius-search phase, we invoke this method with a fixed maximum capacity of photons, a value that we independently tune prior to the rendering phase based on the maximum number of gathered photons recorded with OptiX. We observed in experiments that al-

Figure 8. Example renderings that demonstrate our approach in closed environments of arbitrary light complexity. From left to right: glass, pool, bathroom and fireplace. All scenes were rendered with path length equal to five, for both eye and light sensors.

locating a tight array effectively minimizes potential overheads induced by memory incoherence. Furthermore, we set the gathering method for the FLANN invocations to *Flann_undefined*, which forces FLANN to decide internally whether to employ an iterative or a heap-based update strategy during the radius search. All experiments use path segments of length five from both the eye sensor and the lights, in order to capture every dominant global-illumination effect. For every scene, the rendering resolution is 1920×1080 and the photon batch size per iteration is set to 500K. Figure 8 presents the example scenes used in our tests.

In Figure 9, we investigated the impact of employing different photon-batch sizes. Specifically, we measured the performance of construction and gathering for photon batches between 100K and 1000K emitted per frame, averaged over 50 frames. We did not include tracing performance since this component is handled from OptiX for both cases. Table 1 summarizes the photon-map construction timings with respect to different photon-batch sizes. Naturally, both data structures were negatively affected by the increase of active photons, however, OptiX construction times remained

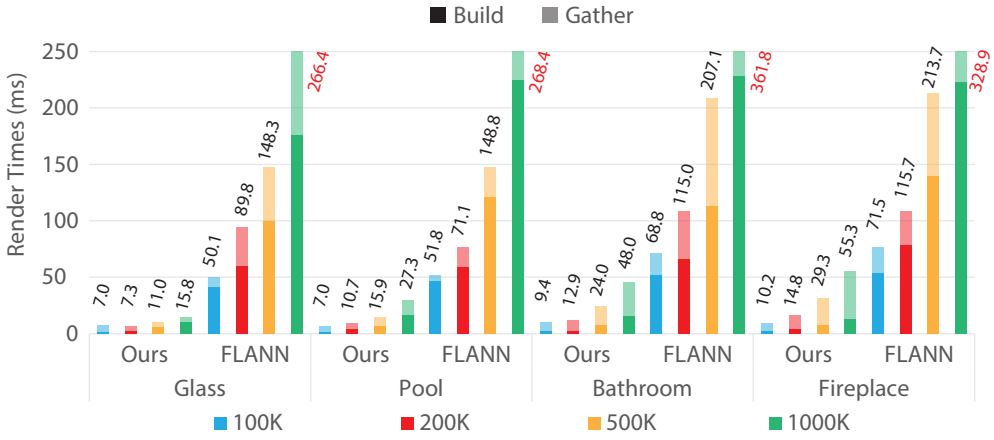

Figure 9. Cumulative performance evaluation of Build and Gather timings averaged over the first 50 iterations, comparing our proposed method and FLANN with increasing size of photon-batch emission for each scene of Figure 8.

	100K	200K	500K	1000K
Build				
Glass	1.3/42.0 (**31.3**×)	2.0/61.9 (**29.7**×)	5.2/99.9 (**19.2**×)	9.4/175.9 (**18.5**×)
Pool	1.9/47.1 (**23.6**×)	3.7/63.3 (**16.7**×)	7.9/123.7 (**15.5**×)	15.1/222.2 (**14.7**×)
Bathroom	2.0/53.1 (**26.1**×)	3.2/68.9 (**21.3**×)	7.6/123.4 (**16.0**×)	14.6/229.9 (**15.7**×)
Fireplace	1.8/56.8 (**31.1**×)	3.7/73.5 (**19.3**×)	6.7/141.6 (**20.9**×)	12.2/220.3 (**18.0**×)
Gather				
Glass	5.6/8.0 (**1.4**×)	5.3/27.9 (**5.2**×)	5.8/48.4 (**8.3**×)	6.3/90.4 (**14.2**×)
Pool	5.8/4.7 (**0.8**×)	6.9/7.8 (**1.1**×)	8.0/25.1 (**3.1**×)	12.2/46.1 (**3.7**×)
Bathroom	7.3/15.6 (**2.1**×)	9.6/46.1 (**4.7**×)	16.3/83.7 (**5.1**×)	33.4/131.9 (**3.9**×)
Fireplace	8.4/14.7 (**1.7**×)	11.0/42.1 (**3.8**×)	22.6/72.1 (**3.1**×)	43.0/108.6 (**2.5**×)

Table 1. Detailed Build and Gather timings of our OptiX implementation vs FLANN on four test scenes using different photon batch sizes. The numbers in the parentheses denote the total improvement over FLANN.

consistently faster than FLANN in every test case, due to the framework's highly-parallel build process as opposed to the less efficient top-down FLANN builder. In the same table, we demonstrate the gathering performance of each framework under the same photon-batch size for every scene. Photon density immediately affects performance for both data structures during the gathering process. Still, our radius-search method performed better in almost every test case. An exception is the pool scene, with photon-batch size equal to 100K, for which performance difference is marginal.Despite the negative impact of the batch size on the relative performance gain of our method, the latter does not drop as fast as in the case of the previous experiments in Sections 4.2 and 4.1. This is mainly due to FLANN's excessive number of gather buffer updates in dense photon regions, in contrast to our method that directly accumulates photon density, without storing intermediate photons.

Finally, it is worth noting that for relatively small photon batches (100K−200K), the total build and gathering operations will consume less than 15ms of the total computation time on every scene. This potentially enables the use of progressive variants of photon mapping for rendering tasks at interactive frame rates.

4.4. Other Experiments

Finally, we re-evaluated the performance of both frameworks using a non-RTX GPU hardware, such the NVIDIA GeForce GTX 1080 with 8GB video memory.

In the case of geometric queries (see Section 4.2), we use an identical configuration for both $k = 1$ and $k = 4$ cases. In Table 2 (left), we present relative performance gains for each GPU card independently for the case of a medium-sized point cloud (350K samples) and a dense one (1M samples). As measurements indicate, the gathering inference time favors our approach in almost all settings. Additionally, the construction performance of the index remains consistently superior.

	k = 1	k = 4
Build		
Metope	8.6×/2.2×	7.5×/2.1×
Hermes	5.8×/1.5×	5.0×/1.6×
Gather		
Metope	2.8×/1.1×	1.5×/0.9×
Hermes	2.0×/1.5×	1.4×/1.2×

	200K	1000K
Build		
Bathroom	19.3×/4.4×	18.0×/4.3×
Fireplace	21.3×/5.1×	15.7×/4.9×
Gather		
Bathroom	4.7×/4.5×	3.9×/3.5×
Fireplace	3.8×/4.4×	2.5×/3.3×

Table 2. Relative performance gain of Build and Gather stages for geometry-processing tasks (left) and progressive photon mapping (right) of our OptiX implementation vs FLANN on an RTX 2080 Ti (red) and a GTX 1080 (blue).

Similarly, in the progressive photon-mapping task (see Section 4.3) and under the same configuration, we measured the performance in two scenes with complex illumination effect such as the bathroom and the fireplace. In Table 2 (right), we show the relative performance gain for low-density and high-density photon-batch sizes (200K and 1M) in which our approach remains superior in all cases.

Due to the underlying hardware improvements specifically targeting hierarchical data-structure construction on the Turing micro-architecture, the corresponding Build-stage gain is significantly higher on the RTX card. However, the relative gain of the Gather stage is not significantly larger on the RTX compared to the non-RTX card. This is to be expected, since the queries do not utilize the triangle-intersection hardware of the former and only take advantage of generic improvements in the newer architecture.

5. Conclusion

In this work, we proposed a mapping of the general radius-search task to the ray-tracing paradigm. Our approach outperforms a GPU implementation of FLANN, a widely used kd-tree option, in all but the most extreme scenarios. Even in those cases, by dramatically reducing the construction time of the supporting spatial-queries index due to the use of highly-optimized ray-tracing acceleration data-structure builders, our approach can quickly close the negative performance gap that might be induced from searching invocations. The superior performance combined with the implementation simplicity, makes our approach a suitable and elegant replacement for any intensive application on the image synthesis or geometric-processing context. The fast acceleration data-structure build time also makes our approach especially appealing for applications where the sample set is continuously updated, such as progressive photon mapping and incremental, live 3D scanning.

The main limitation of our approach is that it cannot be used with arbitrarily large radii, e.g., for performing unbounded k-nearest neighborhood queries. This would imply that each sample bounding-box volume is occupying most of the available total

space with the worst case being the total sample space itself. This effectively eliminates any possible object-based splitting strategy, resulting in a single flat 1-level tree hierarchy that would render our approach completely ineffective. Still, in most practical applications, this extreme case is seldom required, as most neighborhood queries impose some sensible limit beyond which samples returned are deemed invalid anyway. Another, more subtle potential limitation is the inherent inability of this method to shrink the radius parameter during the search of k-nearest samples, since the radius is used for the determination of the node bounds of the hierarchical data structure. This may lead to queries in dense areas searching for a limited number of point samples performing redundant operations, compared to a typical kd-tree. However, throughout our experiments, and especially in the photon-mapping tests, we did not encounter any noticeable performance degradation.

Acknowledgments

The Bathroom scene was based on a model from https://www.cgtrader.com/ and the Fireplace was downloaded from McGuire's Computer Graphics Archive [McGuire 2017]. The remaining scenes were created by the authors. This research is co-financed by Greece and the European Union (European Social Fund-ESF) through the Operational Programme "Human Resources Development, Education and Lifelong Learning 2014-2020" in the context of the project "Modular Light Transport for Photorealistic Rendering on Low-power Graphics Processors" (5049904).

Appendix: Surface Heuristic as an Upper Bound to the Volume Heuristic

In this section, we discuss and establish theoretical guarantees about the final tree quality of a SAH-optimized tree builder for our radius-search task. The SAH function is defined as

$$\mathbf{C}_{\text{sah}}(T) = C_i \sum_{n \in \mathbf{I}} \frac{SA(n)}{SA(\text{root})} + C_l \sum_{l \in \mathbf{L}} \frac{SA(n)}{SA(\text{root})} + C_k \sum_{l \in \mathbf{L}} \frac{SA(l)}{SA(\text{root})} N(l), \tag{3}$$

where $\mathbf{C}_{\text{sah}}(T)$ is the expected cost of the constructed tree T and $\mathbf{I}$, $\mathbf{L}$ account for the set of interior and leaf nodes, respectively. For any given node n of T, function $SA(n)$ calculates the surface area of an AABB and if $n \in \mathbf{L}$, function $N(n)$ returns the number of primitives records enclosed by it. The quantity root stands for the bounding box of the entire scene. The ratio of the surface areas is the conditional probability that an un-occluded ray starting from the scene root will also hit the node. Finally, C_i, C_l, and C_k measure the intersection cost of an interior and leaf node as well as the primitive at a leaf, respectively. Typically, $C_k \geq C_i$ and $C_l = 0$. Similarly, the volume heuristic (VH) cost function is

$$\mathbf{C}_{vh}(T) = C_i \sum_{n \in \mathbf{I}} \frac{V(n)}{V(\text{root})} + C_k \sum_{l \in \mathbf{L}} \frac{V(l)}{V(\text{root})} N(l), \tag{4}$$

where $V(\cdot)$ is the volume of a node. The rest of the parameters are identical to the surface-area heuristic cost.

We want to prove the following statement :

$$\mathbf{C}_{\text{sah}}(T) \geq \mathbf{C}_{vh}(T). \tag{5}$$

Let S be an arbitrary set of samples $\mathbf{s}_j$, each with an AABB. Let also T be the set of all possible binary trees formed for the hierarchical storage of the above sample bounds. For any node n in any $T \in T$ the surface area and volume are

$$SA(n) = 2(wh + hd + wd),$$
$$V(n) = whd,$$

where w, h and d corresponds to the width, height and depth of AABB, respectively. Given also the root-node dimensions $\tilde{w}, \tilde{h}, \tilde{d}$, with the obvious property $\tilde{d} \geq d$, $\tilde{h} \geq h$, $\tilde{w} \geq w$, the following relations hold:

$$\tilde{d} \geq d \Leftrightarrow wh(\tilde{w}\tilde{h}\tilde{d}) \geq \tilde{w}\tilde{h}(whd),$$
$$\tilde{h} \geq h \Leftrightarrow wd(\tilde{w}\tilde{h}\tilde{d}) \geq \tilde{w}\tilde{d}(whd),$$
$$\tilde{w} \geq w \Leftrightarrow hd(\tilde{w}\tilde{h}\tilde{d}) \geq \tilde{h}\tilde{d}(whd).$$

Summing up each side of the above inequalities we get

$$(wh + wd + hd)(\tilde{w}\tilde{h}\tilde{d}) \geq (\tilde{w}\tilde{h} + \tilde{w}\tilde{d} + \tilde{h}\tilde{d})(whd) \Leftrightarrow$$
$$\frac{wh + wd + hd}{\tilde{w}\tilde{h} + \tilde{w}\tilde{d} + \tilde{h}\tilde{d}} \geq \frac{whd}{\tilde{w}\tilde{h}\tilde{d}} \Leftrightarrow \frac{SA(n)}{SA(root)} \geq \frac{V(n)}{V(root)}.$$

Since every component of the cost models in Equations (3) and (4) is non-negative and C_i, C_k, and $N(l)$ are identical in both cost models, it is straightforward to show that Equation (5) holds, which now concludes our proof.

References

BENTLEY, J. L. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM 18*, 9 (Sept.), 509–517. URL: https://doi.org/10.1145/361002.361007. 27

BESL, P. J., AND MCKAY, N. D. 1992. A method for registration of 3-D shapes. *IEEE Trans. Pattern Anal. Mach. Intell. 14*, 2 (Feb.), 239–256. URL: https://doi.org/10.1109/34.121791. 27

DOMINGUES, L. R., AND PEDRINI, H. 2015. Bounding volume hierarchy optimization through agglomerative treelet restructuring. In *Proceedings of the 7th Conference on High-Performance Graphics*, Association for Computing Machinery, New York, NY, USA, HPG '15, 13–20. URL: https://doi.org/10.1145/2790060.2790065. 27

FABIANOWSKI, B., AND DINGLIANA, J. 2009. Interactive global photon mapping. *Computer Graphics Forum 28*, 4, 1151–1159. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2009.01492.x. 28, 29

GEORGIEV, I., KŘIVÁNEK, J., DAVIDOVIČ, T., AND SLUSALLEK, P. 2012. Light transport simulation with vertex connection and merging. *ACM Trans. Graph. 31*, 6 (Nov.). URL: https://doi.org/10.1145/2366145.2366211. 26

GUENNEBAUD, G., AND GROSS, M. 2007. Algebraic point set surfaces. *ACM Trans. Graph. 26*, 3 (July). URL: https://doi.org/10.1145/1276377.1276406. 38

HACHISUKA, T., OGAKI, S., AND JENSEN, H. W. 2008. Progressive photon mapping. *ACM Trans. Graph. 27*, 5 (Dec.). URL: https://doi.org/10.1145/1409060.1409083. 30, 34, 39

HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. 1992. Surface reconstruction from unorganized points. *SIGGRAPH Comput. Graph. 26*, 2 (July), 71–78. URL: https://doi.org/10.1145/142920.134011. 27, 38

JENSEN, H. W. 1996. Global illumination using photon maps. In *Proceedings of the Eurographics Workshop on Rendering Techniques '96*, Springer-Verlag, Berlin, Heidelberg, 21–30. 26, 27

KARRAS, T., AND AILA, T. 2013. Fast parallel construction of high-quality bounding volume hierarchies. In *Proceedings of the 5th High-Performance Graphics Conference*, Association for Computing Machinery, New York, NY, USA, HPG '13, 89–99. URL: https://doi.org/10.1145/2492045.2492055. 27

KNAUS, C., AND ZWICKER, M. 2011. Progressive photon mapping: A probabilistic approach. *ACM Trans. Graph. 30*, 3 (May). URL: https://doi.org/10.1145/1966394.1966404. 39

KNOLL, A., MORLEY, R. K., WALD, I., LEAF, N., AND MESSMER, P. 2019. Efficient particle volume splatting in a ray tracer. In *Ray Tracing Gems: High-Quality and Real-Time Rendering with DXR and Other APIs*. Apress, Berkeley, CA, 533–541. URL: https://doi.org/10.1007/978-1-4842-4427-2_29. 28

KŘIVÁNEK, J., GAUTRON, P., PATTANAIK, S., AND BOUATOUCH, K. 2008. Radiance caching for efficient global illumination computation. In *ACM SIGGRAPH 2008 Classes*, Association for Computing Machinery, New York, NY, USA, SIGGRAPH 08. URL: https://doi.org/10.1145/1401132.1401228. 27

LAUTERBACH, C., GARLAND, M., SENGUPTA, S., LUEBKE, D., AND MANOCHA, D. 2009. Fast BVH construction on GPUs. *Computer Graphics Forum 28*, 2, 375–384. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2009.01377.x. 27, 28

MA, V. C. H., AND MCCOOL, M. D. 2002. Low latency photon mapping using block hashing. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, Eurographics Association, Aire-la-Ville, Switzerland, HWWS '02, 89–99. URL: http://dl.acm.org/citation.cfm?id=569046.569059. 27

MACDONALD, D. J., AND BOOTH, K. S. 1990. Heuristics for ray tracing using space subdivision. *Vis. Comput. 6*, 3 (May), 153–166. URL: https://doi.org/10.1007/BF01911006. 29

MCGUIRE, M., 2017. Computer graphics archive, July. URL: https://casual-effects.com/data. 43

MEAGHER, D. 1982. Geometric modeling using octree encoding. *Computer Graphics and Image Processing 19*, 2, 129 –147. URL: http://www.sciencedirect.com/science/article/pii/0146664X82901046. 27

MUJA, M., AND LOWE, D. G. 2009. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP International Conference on Computer Vision Theory and Applications*, INSTICC Press, 331–340. 34

PARKER, S. G., BIGLER, J., DIETRICH, A., FRIEDRICH, H., HOBEROCK, J., LUEBKE, D., MCALLISTER, D., MCGUIRE, M., MORLEY, K., ROBISON, A., AND STICH, M. 2010. Optix: A general purpose ray tracing engine. *ACM Trans. Graph. 29*, 4 (July). URL: https://doi.org/10.1145/1778765.1778803. 32

RUSINKIEWICZ, S., AND LEVOY, M. 2001. Efficient variants of the ICP algorithm. In *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*. IEEE, New York, NY, USA, 145–152. URL: https://ieeexplore.ieee.org/document/924423. 32

RUSU, R. B., AND COUSINS, S. 2011. 3D is here: Point Cloud Library (PCL). In *2011 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, New York, NY, USA, May 9-13, 1–4. URL: https://doi.org/10.1109/ICRA.2011.5980567. 27

SRIWASANSAK, J., GRUSON, A., AND HACHISUKA, T. 2018. Efficient energy-compensated VPLs using photon splatting. *Proc. ACM Comput. Graph. Interact. Tech. 1*, 1 (July). URL: https://doi.org/10.1145/3203189. 26

STÜRZLINGER, W., AND BASTOS, R. 1997. Interactive rendering of globally illuminated glossy scenes. In *Proceedings of the Eurographics Workshop on Rendering Techniques '97*. Springer-Verlag, Berlin, Heidelberg, 93–102. URL: https://link.springer.com/chapter/10.1007/978-3-7091-6858-5_9. 29

VORBA, J., HANIKA, J., HERHOLZ, S., MÜLLER, T., KŘIVÁNEK, J., AND KELLER, A. 2019. Path guiding in production. In *ACM SIGGRAPH 2019 Courses*, Association for Computing Machinery, New York, NY, USA, SIGGRAPH '19. URL: https://doi.org/10.1145/3305366.3328091. 26

WALD, I., GÜNTHER, J., AND SLUSALLEK, P. 2004. Balancing considered harmful - Faster photon mapping using the voxel volume heuristic -. *Computer Graphics Forum 23*, 3, 595–603. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2004.00791.x. 27

WALD, I., USHER, W., MORRICAL, N., LEDIAEV, L., AND PASCUCCI, V. 2019. RTX beyond ray tracing: Exploring the use of hardware ray tracing cores for tet-mesh point location. In *High-Performance Graphics - Short Papers*, M. Steinberger and T. Foley, Eds. The Eurographics Association, Aire-la-Ville, Switzerland. URL: https://diglib.eg.org/handle/10.2312/hpg20191189. 28

WANG, Y., KHIAT, S., KRY, P. G., AND NOWROUZEZAHRAI, D. 2019. Fast non-uniform radiance probe placement and tracing. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, Association for Computing Machinery, New York, NY, USA, I3D 19. URL: https://doi.org/10.1145/3306131.3317024. 27

WARD, G. J., RUBINSTEIN, F. M., AND CLEAR, R. D. 1988. A ray tracing solution for diffuse interreflection. *SIGGRAPH Comput. Graph. 22*, 4 (June), 8592. URL: https://doi.org/10.1145/378456.378490. 26, 27

ZELLMANN, S., WEIER, M., AND WALD, I. 2020. Accelerating force-directed graph drawing with RT cores. In *IEEE Visualization (Short Papers)*. IEEE, New York, NY, USA. URL: https://virtual.ieeevis.org/paper_s-short-1027.html. 28

ZHANG, Z. 1994. Iterative point matching for registration of free-form curves and surfaces. *Int. J. Comput. Vision 13*, 2 (Oct.), 119152. URL: https://doi.org/10.1007/BF01427149. 27

ZHOU, K., HOU, Q., WANG, R., AND GUO, B. 2008. Real-time KD-tree construction on graphics hardware. *ACM Trans. Graph. 27*, 5 (Dec.). URL: https://doi.org/10.1145/1409060.1409079. 27

Index of Supplemental Materials

The full source code of this work can be found at https://github.com/cgaueb/fast_radius_search

Author Contact Information

Iordanis Evangelou
Department of Informatics
Athens University of Economics
& Business
76 Patission St
Athens, 10434 Greece
iordanise@aueb.gr

Georgios Papaioannou
Department of Informatics
Athens University of Economics
& Business
76 Patission St
Athens, 10434 Greece
gepap@aueb.gr

Konstantinos Vardis
Department of Informatics
Athens University of Economics
& Business
76 Patission St
Athens, 10434 Greece
kvardis@aueb.gr

Andreas A. Vasilakis
Department of Informatics
Athens University of Economics
& Business
76 Patission St
Athens, 10434 Greece
abasilak@aueb.gr

Iordanis Evangelou, Georgios Papaioannou, Konstantinos Vardis, Andreas A. Vasilakis, Fast Radius Search using Bounding Volume Hierarchies, *Journal of Computer Graphics Techniques (JCGT)*, vol. 10, no. 1, 25–48, 2021
http://jcgt.org/published/0010/01/02/

Received:	2020-07-07		
Recommended:	2020-11-24	Corresponding Editor:	Eric Haines
Published:	2021-02-05	Editor-in-Chief:	Marc Olano