3DPro: Querying Complex Three-Dimensional Data with Progressive Compression and Refinement

Dejun Teng
Stony Brook University

Furqan Baig
Stony Brook University

Vo Hoang
Pitney Bowes Inc.

Yanhui Liang
Waymo LLC

Jun Kong
Georgia State University

Fusheng Wang
Stony Brook University

ABSTRACT

Large-scale three-dimensional spatial data has gained increasing attention with the development of self-driving, mineral exploration, CAD, and human atlases. Such 3D objects are often represented with a polygonal model at high resolution to preserve accuracy. This poses major challenges for 3D data management and spatial queries due to the massive amounts of 3D objects, e.g., trillions of 3D cells, and the high complexity of 3D geometric computation. Traditional spatial querying methods in the Filter-Refine paradigm have a major focus on indexing-based filtering using approximations like minimal bounding boxes and largely neglect the heavy computation in the refinement step at the intra-geometry level, which often dominates the cost of query processing. In this paper, we introduce *3DPro*, a system that supports efficient spatial queries for complex 3D objects. 3DPro uses progressive compression of 3D objects preserving multiple levels of details, which significantly reduces the size of the objects and has the data fit into memory. Through a novel Filter-Progressive-Refine paradigm, 3DPro can have query results returned early whenever possible to minimize decompression and geometric computations of 3D objects in higher resolution representations. Our experiments demonstrate that 3DPro outperforms the state-of-the-art 3D data processing techniques by up to an order of magnitude for typical spatial queries.

1 INTRODUCTION

Spatial data is exploding because of the wide availability of GPS devices, volunteered geographic information, and the development of high-resolution high-throughput imaging instruments. In particular, 3D spatial data has been used in various industrial applications such as mineral exploration [51], 3D GIS [22], urban planning [12], and 3D mapping and navigation [68]. Extreme-scale 3D data is also produced from human tissues. Indeed, the NIH Human Biomolecular Atlas Program (HuBMAP) [16] promotes the development of technologies for 3D mapping of the human body with extreme resolution at a sub-cellular level, to derive trillions of 3D cells to be mapped and queried, together with surrounding vasculature structures and molecular level objects. The Human Tumor Atlas Network (HTAN) [23] aims to construct 3D atlases of the dynamic cellular, morphological, and molecular features of human cancers as they evolve. For example, with 3D pathology imaging [36, 39], 3D nuclei, cells, and blood vessels can be reconstructed as 3D geometries representing the fine structures of the human body for disease studies and computer-aided diagnosis.

3D data provides a more accurate representation compared with 2D data projected from objects in 3D space, but comes with much higher complexity, in terms of both shapes and surface-based 3D modeling. Example 3D structures in human tissues include nuclei, cells, vessels, among others. Nuclei and cells have relatively simpler shapes (Fig. 4), and a blood vessel is bifurcated and has tree-like structures (Fig.1). For representing 3D objects, polygonal modeling [52] is the most commonly used approach for modeling 3D objects by representing or approximating their surfaces using polygonal meshes because of its advantage of the flexibility and quick rendering. While higher resolution represents a better description of the original object and can yield more precise query results, it will require much higher computation and storage resources for both query processing and rendering. Another major challenge is the explosion of 3D data. For example, a single human tissue block may contain hundreds of millions of 3D objects. The complexity of object surfaces and the large scale of 3D data lead to heavy-duty computations in 3D spatial query processing when the intersection between 3D objects needs to be checked or the distance needs to be calculated.

Traditional spatial querying methods in the Filter-Refine paradigm [11] use approximations of spatial objects to produce a set of candidates in the filtering step, and examine the candidates concerning its exact geometry to generate exact answers in the refinement step. This paradigm has a major focus on indexing-based filtering using minimal bounding boxes (MBBs) to effectively minimize the number of objects to be processed, and largely ignore the high complexity inside the 3D geometry. Indeed, the refinement step can dominate query cost for many query types [1, 8, 63]. Each 3D object can be represented by hundreds to thousands of mesh faces, which can make refinement a dominant step in 3D query processing. For instance, it takes PostGIS [49], the most popular open-source spatial database management system, more than one hour to compute the distance between two complex 3D objects, each with 30,000 surface faces. As shown in Section 6, the refinement step takes more than 99% of the execution time for all tests.

Various approaches have been proposed to mitigate the heavy computation in the refinement step at the intra-geometry level. The first approach is to build spatial indexes like R-tree [27, 41, 49, 50], Axis-Aligned Bounding Box tree (AABB-tree) [13], and Orientable Bounding Box tree (OBB-tree) [26] on the primitives of the polyhedrons to facilitate the spatial relationship determinations between individual objects. The second approach is to partition the complex objects into multiple simple sub-objects in regular shapes with fewer surface faces [4, 9, 26]. Approximating a complex object with the MBBs of all its sub-objects will first bring a better fitting of its shape thus a better filtering performance in the filtering step than approximating it with a single MBB. Besides, since only a subset of sub-objects needs to be evaluated in the refinement step, the total amount of geometric

computations will be reduced. The third approach is to accelerate the geometric computation with massively parallel computing devices like GPU [51]. These approaches either focus on better intra-geometry level filtering or parallel computing, and none of them reduce the complexity of the 3D objects.

In this paper, we take an alternative approach to reduce the complexity of 3D objects by representing a 3D object (modeled with polygonal meshes) with multiple levels of resolutions with compression, where lower resolution representation is through removing surface meshes. This will make it possible to use lower resolution representation for possible early filtering and progress to higher resolution representation only if needed. i.e., we will have a new paradigm of Filter-Progressive-Refine paradigm for spatial query processing. With filtering possible at a lower resolution, the complexity of geometric computation can be much reduced. To achieve this goal, we propose a novel multi-level progressive 3D compression method, designed to guarantee that the compressed objects are certain types of approximations of the original objects. This will support returning correct results even with compressed data whenever possible. Based on this, we further propose novel progressive refinement for query processing, which starts from lightweight low-resolution representation for possible early return of results. A system *3DPro* is implemented with progressive refinement to support highly efficient and accurate spatial queries for complex three-dimensional objects. The contributions can be summarized as:

- We introduce a *progressive protruding-vertex pruning mesh compression* method which reduces object complexity progressively to create multi-resolution 3D representations;
- We provide a Filter-Progressive-Refine paradigm based on progressive compression to minimize geometric computation, which can provide early returns of accurate results from lower resolution whenever possible;
- We take a memory-centered approach for data management and querying processing to mitigate I/O cost;
- We implement and evaluate our approach on three representative 3D spatial queries: intersection, within, and nearest neighbor search;
- We implement three existing techniques and prove that the proposed Filter-Progressive-Refine paradigm can co-exist with all of them to achieve optimized 3D spatial query.

The rest of the paper is organized as follows. The background on 3D data representation and compression is introduced in Section 2. The novel technique to facilitate 3D spatial compression by handling the complexity of 3D objects is described in Section 3. Progressive 3D query processing is discussed in Section 4. Section 5 further provides implementation details of 3DPro. The performance of 3DPro is evaluated in Section 6, followed by related works and conclusion in Section 7 and Section 8 respectively.

2 BACKGROUND

2.1 3D Data Representation

In 3D space, physical objects are represented by different 3D models [37]. Among them, polygonal modeling represents or approximates the surfaces of a 3D object using polygonal meshes [52]. In this paper, we represent a 3D object as a polyhedron, a polygonal mesh that encloses a bounded region of space and has no unnecessary edge junctions. A polyhedron consists of a collection of *vertices*, *edges*, and *faces*. All the faces are orientable - a face will have an outer side and an inner side. In this paper, the i^{th} vertex of a polyhedron is denoted as v_i, and each face is represented with the vertices in counter-clockwise order, such as $v_0v_1v_2$. Therefore, the outer side of a face can be determined with the right-hand rule.

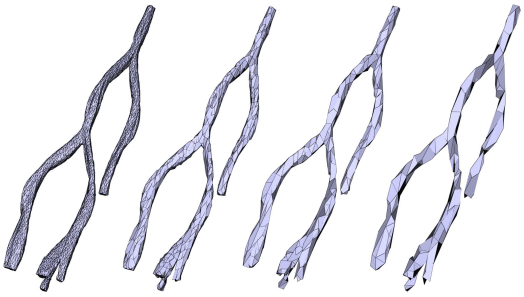


Figure 1: An object represented as polyhedrons in different levels of details (LODs)

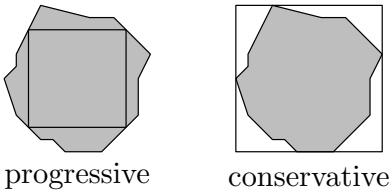


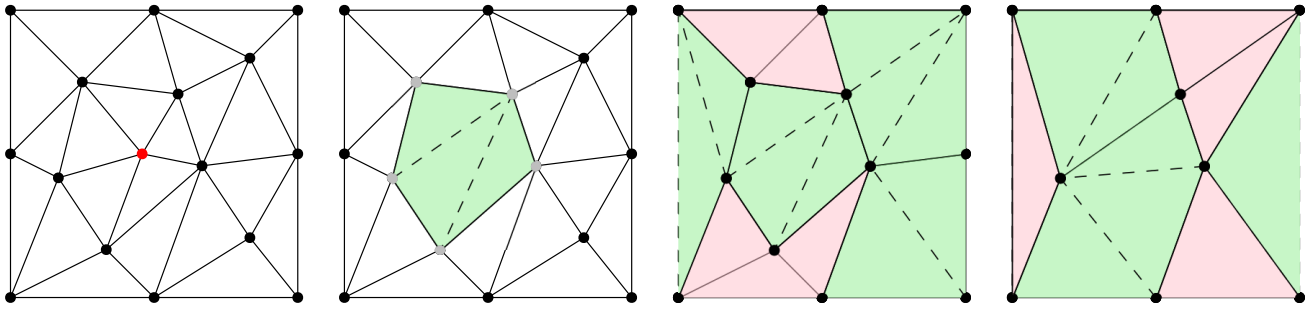
Figure 2: Two types of spatial object approximations

2.2 Spatial Object Approximation

A common method to improve the spatial query performance is to use object approximations to filter out as many objects as possible [9]. As shown in Fig. 2, there are two typical types of approximations: progressive approximation covers a subset region of the original object, and conservative approximation covers a superset region of the original object. These approximations can be utilized to facilitate queries based on following properties: 1) Two objects will intersect if their progressive approximations intersect. Meanwhile, two objects will not intersect if their conservative approximations do not intersect. 2) The distance between two objects is smaller than the distance between their progressive approximations but larger than the distance between their conservative approximations. An approximation could be a box, an ellipse, an ellipsoid, or another polyhedron with a lower resolution.

2.3 3D Object Representation with Multiple Levels of Details

3D spatial compression: A 3D object can be represented with 3D polyhedrons in different resolutions, or *levels of details* (LOD) [37]. A higher level of detail implies more faces on the surface, thus a more precise representation of the 3D object. A higher LOD also means more memory and disk space for storage and more computation resources for processing. Spatial compression is a common method to reduce the resolution of representing spatial objects. In 2D space, spatial compression can be achieved with line simplification by removing selected points such as the *Douglas-Peucker* algorithm [19, 60]. Similarly, in 3D space, compression can be conducted on 3D spatial objects by collapsing faces, edges, and vertex pairs, or removing vertices [37]. Various compression methods are developed to improve the compression rate and distortion rate [29, 32, 38, 46, 47, 59]. Next, we briefly



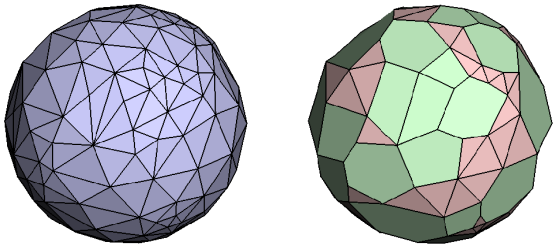
(a) Projection of the original polyhedron

(b) One vertex and five faces are removed

(c) The polyhedron after one round of decimation

(d) The polyhedron after two rounds of decimation

Figure 3: Simplifying polyhedron surface by removing vertices and faces



(a) Original polyhedron

(b) Decimated polyhedron

Figure 4: One round of decimation conducted on a polyhedron. Green colored faces with a vertex are removed.

introduce the typical steps of a classical method on compressing 3D spatial objects by removing vertices [38].

Fig. 3a shows the projection of part of a polyhedron to 2D space. Initially, an arbitrary vertex is selected for removal. As is shown in Fig. 3b, the vertex and all the faces connected to it are removed. As a result, a hole is generated, which is then filled with new triangles to make the mesh manifold. By removing the vertex, the total number of faces is reduced by 2. Meanwhile, all the vertices sharing edges with the removed vertex will be marked as *irremovable* and all the *irremovable* vertices should not be removed in this round of mesh simplification. One vertex after another, one round of simplification is conducted by removing all the vertices except the *irremovable* ones. Fig. 3c shows the surface of the polyhedron after one round of simplification. It shows that 4 vertices are removed and the number of faces is reduced by 7. One round of simplification is also named one round of decimation [38]. For each round of decimation, the removed vertices are compressed and stored together for decoding, thus the compression process is invertible.

Progressive compression: By avoiding the removal of two vertices sharing the same edge, all parts of the surface of a polyhedron can be evenly simplified in one round of decimation. Therefore, we say that the polyhedron representing the object after one or multiple rounds of decimation as one LOD. Fig. 4 shows the original polyhedron and the polyhedron after one round of decimation. After one round of decimation, all the vertices left are reset as *removable* and another round of decimation can be further conducted. Fig. 3d shows the polyhedron after two rounds of decimation. Another 3 vertices are removed and the number of faces is further reduced by 5. As the compression is conducted round by round, it is called *progressive compression*.

Progressive compression is attractive in supporting efficient spatial querying for representing multi-LODs of a 3D object in a single compressed format, for example, progressive polygon mesh compression (PPMC) [38]. With PPMC, the compressed object can be fully decoded to reconstruct the original 3D object without loss, or a representation at a specific LOD can be decoded for an approximated representation. Decoding to lower LOD representation is less expensive as the decoding is conducted progressively from a lower LOD to the next level, thus it is desirable to support *progressive queries*: determining spatial relationships at lower LOD for early return of query result. This has to rely on the properties of querying with approximations introduced in Section 2.2. However, as will be discussed in Section 3.2, the output polyhedrons compressed with the PPMC compression are neither progressive nor conservative approximations of the original objects, such that none of those two properties holds. **In this paper, we propose a new progressive compression method which guarantees that the compressed polyhedron is a progressive approximation of the original polyhedron, and makes it possible to support precise queries with querying representations at low LODs.**

2.4 Representative 3D Objects and Queries

While 3D objects vary dramatically between applications, in this paper, we mainly focus on two representative 3D structures: regular shape 3D geometries, e.g., nuclei/cells as shown in Fig. 4, and bifurcated geometries, e.g., blood vessels as shown in Fig.1. Nuclei and vessels are the two most representative 3D structures from human tissues in digital pathology [17]. Nuclei are small 3D objects but come in vast amounts and relatively simple shapes. Vessels are large complex bifurcated objects but the number is smaller. Note that our methods are generic and are not tissue objects specific. In this paper, we assume that 3D objects are represented as 3D polyhedrons and have already been extracted through image registration, segmentation, and reconstruction from original serials of 2D images [33, 54].

3D digital pathology holds significant potential to enhance the study of both normal and disease processes, particularly, those involving structural changes and those in which the spatial relationship of disease features is important [25, 39]. Generally speaking, spatial queries can be categorized into two sets. The first set of queries is the ones to retrieve features from spatial objects, for instance, queries to get the perimeter/area/volume of an object or to get the distance between two objects. The second set of queries are the ones to determine the spatial relationship between spatial objects, such as intersection, containment, within

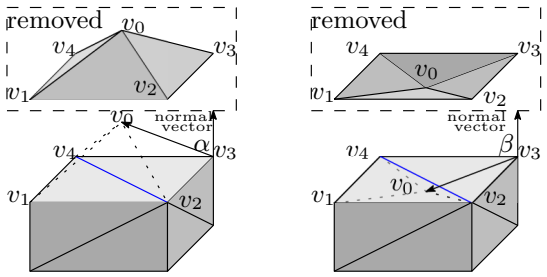
or nearest neighbor relationships. This paper focuses on the second set of queries which determines the spatial relationship between objects. More specifically, we will focus on three typical queries: intersection, within, and nearest neighbor, and other query types can be implemented with a similar approach.

Spatial join is a common complex query type in both geospatial applications and 3D digital pathology. Given two datasets D_1 and D_2 of 3D objects, *spatial join* will return a set S of objects from D_2 for each object o in D_1, such that all those retrieved objects in S meet one of the spatial relationships mentioned above with o [8]. Note that the nearest neighbor spatial join is also known as All Nearest Neighbor query, or ANN query [69]. For example, we may want to identify the closest blood vessels for each nucleus.

3 PROGRESSIVE PROTRUDING-VERTEX PRUNING MESH COMPRESSION

In this section, we propose a novel progressive compression method to overcome the limitation of the PPMC method [38]. Compared with PPMC, the new compression method offers additional information in achieving accurate querying with polyhedrons representing objects in low LODs. Next, we introduce "protruding vertex" and explain how the new method guarantees the "progressiveness" of the lower LOD polyhedrons by only removing the "protruding vertices" during the encoding process.

3.1 Protruding Vertex



(a) Protruding vertex **(b) Recessing vertex**

Figure 5: Protruding and recessing vertices

Removing different vertices from a polyhedron during the decoding process can make different impacts on the original polyhedron. As shown in Fig. 5a, after a vertex v_0 is removed during the encoding process of progressive compression, all the faces v_0 connects are also removed. New edge v_2v_4 and two triangle faces $v_1v_2v_4$ and $v_2v_3v_4$ are added to enclose the space bounded by the new polyhedron. v_o, together with each of those added faces will form a tetrahedron, thus removing one vertex from the original polyhedron in the encoding process equals removing multiple tetrahedrons. Removing one of those tetrahedrons will be either cutting off a solid part from the original polyhedron or filling a "pit" on the surface of the original polyhedron. Whether a solid part will be cut or a "pit" will be filled can be determined by the intersection angle between the normal vector of the newly added face and a vector pointing from a random point on the newly added face to the removed vertex. If the intersection angle is acute, removing the tetrahedron will cut off a solid part and make the polyhedron slimmer (angle α in Fig. 5a). If the intersection angle is obtuse, removing the tetrahedron will fill a "pit" and make the polyhedron thicker (angle β in Fig. 5b). Otherwise, the tetrahedron is invalid, and removing it will have no impact on the polyhedron. If removing one vertex can only cut off solid parts from the original polyhedron or have no impact, we name the vertex as *"protruding" vertex* (Fig. 5a), and all other vertices as *"recessing" vertices* (Fig. 5b).

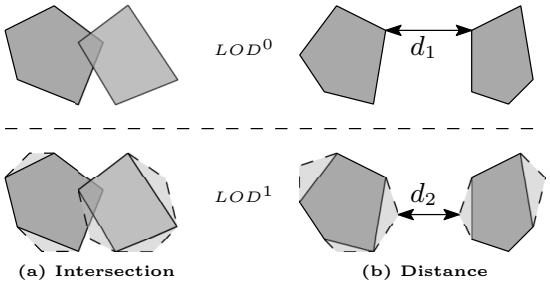


Figure 6: Properties of querying multiple LODs

3.2 Progressive Protruding-Vertex Pruning

It can be summarized that in the PPMC compression process, removing vertices can either cut a solid part (removing protruding vertex) or fill a "pit" (removing recessing vertex), and there is no guarantee that the compressed polyhedron will cover a subset or superset of the original object. By profiling the real-world dataset, we find that the majority of the vertices (e.g. 92% of the vertices in our tests) are protruding. If all the vertices removed during the encoding process are protruding, the compression process will always prune the object to make it slimmer. We call such compression *Progressive Protruding-Vertex Pruning compression*, or *PPVP compression* for short. As a result, a representation of an object at a lower LOD compressed with the PPVP compression is always a *progressive* approximation (Fig. 2) of the representation at a higher LOD. Based on these, those two approximation querying properties can be extended as:

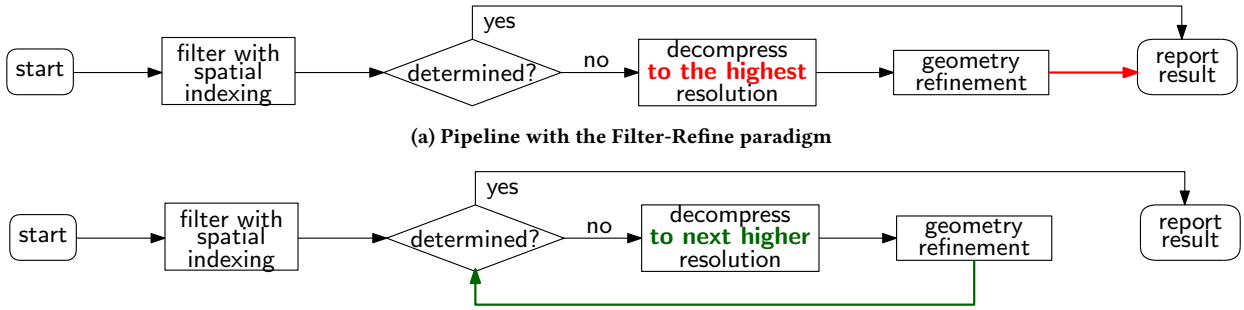
- If two objects intersect with each other at a lower LOD, they must also intersect at a higher LOD. (Fig. 6a)
- The distance of two objects at a lower LOD must be longer than or equal to that at a higher LOD. (Fig. 6b, $d_1 > d_2$)

The distance of two objects mentioned in this paper equals the distance of two closest points at the surface of those two objects at their highest LODs. Hausdorff distance [30] focuses on comparing shapes and patterns to measure similarity, and will be out of the scope of this paper.

4 FILTER-PROGRESSIVE-REFINE PARADIGM

In this section, we introduce a novel *Filter-Progressive-Refine paradigm* (short as FPR paradigm) which consists of a filtering step and a progressive refinement step built on our PPVP compression. The term *progressive* indicates that low LOD polyhedrons are progressive approximations of the original objects; it also indicates that the compression is conducted in multiple rounds, or progressively. Next, we will discuss how the FPR paradigm can be utilized to facilitate three typical 3D spatial queries: intersection, within, and nearest neighbor search [43]. We assume the original 3D objects are compressed based on PPVP and stored in memory, and the indexes are generated and stored in memory.

A comparison between the Filter-Progressive-Refine paradigm with the traditional Filter-Refine paradigm (short as FR paradigm) is shown in Fig. 7. For each object in the target dataset, the candidate objects in the source dataset are retrieved by checking



(a) Pipeline with the Filter-Refine paradigm

(b) Pipeline with the Filter-Progressive-Refine paradigm

Figure 7: Pipelines of conducting queries with the Filter-Refine paradigm and the Filter-Progressive-Refine paradigm

the *spatial indexing* such as *R-tree* in the filtering step of both paradigms. In the refinement step of the traditional Filter-Refine paradigm (Fig. 7a), all the retrieved objects are decoded to the highest LOD for refined spatial relationship determination to get the final result. In the progressive refinement step of the Filter-Progressive-Refine paradigm (Fig. 7b), instead, the candidate objects are decoded from lower LODs to next level LOD and refined progressively until the spatial relationships between the target object and candidates can be determined. Note that the querying methods introduced in this section work for all dimensions as long as the representations in all LODs cover a subset of the original object, or are all progressive approximations.

4.1 Intersection Query

Given a 3D object o and a set of 3D objects D, *intersection query* will return a set S of objects from D, such that o intersects every object in S.

Get intersection candidates by querying the R-tree: As a first step to compute S for an object o, based on the fact that two objects whose MBBs do not intersect are guaranteed not to intersect, intersection detection can be accelerated by using *R-trees* [10, 11]. The objects in D whose MBBs intersect with the MBB of o are pushed into list *candidates* for further evaluation as the input of the refinement step.

Algorithm 1: Determine the objects actually intersect o with the progressively decoded faces

```
   Input  : o-target object, candidates-intersection candidates
   Output : results-list of objects truly intersect o
1  for each lod in lod_list do
2      geom ← decode(o, lod);
3      for each object o' in candidates do
4          geom' ← decode(o', lod);
5          if intersect(geom, geom') = true then
6              results.insert(o');
7              candidates.remove(o');
   /* handle the containment cases                    */
8  for each object o' in candidates do
9      if contain(o.MBB, o'.MBB) = true then
10         v ← a random vertex of o';
11         if contain(o, v) = true then
12             results.insert(o');
       /* test on o' contains o is omitted            */
```

Determining intersection with decoded faces: The second stage is to retrieve objects which actually intersect the target object o from the candidate list (Alg. 1). One common method to evaluate whether a candidate object o' actually intersects object o is to check whether any pairs of faces in o and o' intersect [58]. With this method, o and o' are decoded into their highest LODs and all the face pairs of them are checked until at least one pair of intersected faces is found and it can be concluded that two objects intersect. Otherwise, after knowing there are no intersected faces between two objects, we can further check if they are contained by each other and finally report whether those two objects intersect.

In contrast, with the Filter-Progressive-Refine paradigm, referred objects are decoded and refined progressively from lower LODs in the progressive refinement step. For a certain LOD, object o and all the objects in the *candidates* list are decoded into that LOD and the faces are filled into the proper memory buffers (steps 2,4). Then all the decoded face pairs of o and o' are evaluated one by one (step 5). If any intersected face pairs are encountered, two objects intersecting with each other will be reported (step 6) and the related entry in *candidates* is removed (step 7). Otherwise, the objects are decoded to a higher LOD and the faces checking process is repeated. Since the geometric computation stops once an intersected pair of faces is found even at lower LODs, the intersection detection among two objects can be highly efficient. If no intersecting face pair is found after checking the decoded faces at the highest LOD of a candidate o', we further check if it contains or is contained by o (steps 8-12). If the MBB of o' is contained by the MBB of o, and a random vertex v in o' is contained by o (step 11), we conclude that o' is contained by o thus o' intersect o. The process of checking whether a point is contained by a polyhedron can also be accelerated by the Filter-Progressive-Refine paradigm, but due to the space limit, we will not discuss it in detail here.

4.2 Within Query

Given a 3D object o and a set of 3D objects D, a *within query* will return a set S of objects from D for object o, such that every object in S is within a specified distance of o.

Get within candidates by querying the R-tree: As is denoted in [15], the distance of two objects can be estimated with a *distance range* $r = [MINDIST, MAXDIST]$ calculated with their MBBs, where *MINDIST* equals the distance of their MBBs, and *MAXDIST* equals the length of the diagonal line of the MBB that unions their MBBs. In other words, $r.MINDIST$ and $r.MAXDIST$ represent the infimum and supremum of the possible distances between any random pair of points covered by two MBBs. Therefore, the

distance ranges between the MBBs of tree nodes and objects can be facilitated to determine whether the objects are within the specified distance of one another [48]. A *within* search can be accelerated with the distance ranges and tree-based indexing. While traversing the *R-tree*, a distance range between the MBBs of o and an *R-tree* node n is calculated as r. All the objects n covers can be skipped if $r.MINDIST$ is larger than the specified distance. Otherwise, if $r.MAXDIST$ is smaller or equal to the specified distance, which means that the distances between o to all the objects covered by n must be smaller than the specified distance, thus all the objects covered by n should be reported as within the specified distance of o. At last, if none of them is true and n is a leaf node, the object at n will be pushed into the *candidates* list for further evaluation with the decoded faces.

Algorithm 2: Determine the objects within a specified distance D with decoded faces

Input : o-target object, *candidates*-within candidates
Output : *results*-list of objects truly within D from o

```
1 for lod in lod_list do
2     geom ← decode(o, lod);
3     for each candidate o' in candidates do
4         geom' ← decode(o', lod);
5         d_m ← distance(geom, geom');
6         if d_m ≤ D then
7             results.insert(o');
8             candidates.remove(o')
```

Determine the within relationship with decoded faces: After getting all the within candidate objects for o by checking spatial indexes, a common method to retrieve the objects which are truly within a specified distance of o is to calculate the distances between the candidate objects and o and keep only the one with distance to o smaller than the specified distance [48]. The distance between o and o' is computed by getting the minimum distance of all the face pairs of the polyhedrons which represent objects o and o' respectively at their highest LODs. The distance calculation between triangles in 3D space is well-studied [42], and we inherit the computation method implemented in [34].

However, with the Filter-Progressive-Refine paradigm, the distances between candidate objects and o are calculated with geometries representing those objects at increasing LODs. As shown in Alg. 2, for each LOD *lod*, o and all the candidate objects are firstly decoded to *lod* (steps 2,4). Then the distance between the geometries representing o and each candidate object o' at LOD *lod*, denoted as d_m, is calculated by getting the minimum distance between the face pairs of them (step 5). If d_m is smaller or equal to the specified distance, it is certain that the actual distance between o and o', which equals the distance between the geometries representing them at the highest LOD, must also be smaller or equal to the specified distance as is regulated by the second progressive query property given in Section 3.2, thus we can stop and report that o' is within the specified distance of o (step 7) and o' can be removed from the *candidates* list (step 8). As a result, the within relationship between o and candidate objects can be determined with geometries representing them at lower LODs, thus the heavy load geometric computations needed for processing high LOD geometries can be saved.

4.3 Nearest Neighbor Query

A *nearest neighbor query* on a 3D object o and a set of 3D objects D returns the nearest object in D for object o.

Retrieve nearest neighbor candidates with R-tree: A nearest neighbor search can also be accelerated with tree-based indexing by facilitating distance ranges between MBBs [15]. The distance relationship of objects can be determined even with only the distance ranges between them. For example, a distance with range r_1 is closer than a distance with range r_2 if $r_1.MAXDIST < r_2.MINDIST$. With this fact, *R-tree* can be utilized to facilitate nearest neighbor search [7, 15, 28, 44].

While traversing the *R-tree*, a list *candidates* maintains all the candidate objects which might be the nearest neighbor of o. A global variable *MINMAXDIST* maintains the minimum *MAXDIST* value of the distance ranges between o to all the objects in *candidates*. If the minimum possible distance between an *R-tree* node n to o, denoted as $r_n.MINDIST$, is larger than *MINMAXDIST*, all the objects it covers can be skipped since the distances from o to those objects must be larger than that to any object in the *candidates* list. Otherwise, if $r_n.MINDIST \leq MINMAXDIST$ and n is a leaf node, the object n covers need to be pushed into the *candidates* list. Finally, the *candidates* list contains all the nearest neighbor candidates whose distance ranges to o overlap with each other, which need to be further evaluated with decoded faces to get the one with the minimum distance to o.

Algorithm 3: Determine the nearest neighbor with decoded faces

Input : o-target object, *candidates*-nearest neighbor candidates
Output : The nearest neighbor object of o

```
1  for lod in lod_list do
2      geom ← decode(o, lod);
       /* r is the distance range between o and o'      */
3      for each candidate (o', r) in candidates do
           /* MINMAXDIST keeps decreasing               */
4          if r.MINDIST>MINMAXDIST then
5              candidates.remove((o', r));
6              Continue;
7          geom' ← decode(o', lod);
8          d_m ← distance(geom, geom');
9          r.MAXDIST← d_m;
10         if lod is the highest then
11             r.MINDIST← d_m;
12         if r.MAXDIST<MINMAXDIST then
13             MINMAXDIST←r.MAXDIST;
14     for each pair (o', r) in candidates do
15         if r.MINDIST>MINMAXDIST then
16             candidates.remove((o', r));
17     if candidates.size = 1 then
18         Report candidates[0] as the nearest neighbor;
```

Determine the nearest neighbor with decoded faces: After getting the nearest neighbor candidates for object o, the common method to get the nearest neighbor is to decode o and all the objects in the candidate list to their highest LODs, then get the distances from o to all those decoded objects and pick the one with the minimum distance as the nearest neighbor.

With the Filter-Progressive-Refine paradigm, instead of comparing the faces of polyhedrons representing objects at the highest LOD, the faces of polyhedrons at low LODs can be evaluated first to further reduce the size of the candidate list to avoid heavy geometric computations conducted on polyhedrons at high LODs. As shown in Alg. 3, multiple iterations of refinements are conducted on the polyhedrons. Starting from a low LOD, object o is decoded and the faces of the polyhedron at that LOD are packed into the memory buffer (step 2). At the beginning of each refinement round, each entry in the *candidates* list needs to be reevaluated with the newest value of *MINMAXDIST* as it is continuously decreasing during the processing. The object whose minimum possible distance to o, denoted as $r.MINDIST$, is larger than *MINMAXDIST* will be removed from *candidates* (step 5). Otherwise, the candidate object o' is decoded to current *lod* (step 7), and the minimum distance of the face pairs between o and o', denoted as d_m, is computed (step 8). Since the PPVP compression is prune-only, the distances between objects at lower LODs are always larger than those at higher LODs. Therefore the supremum of the distance range between objects o and o', or $r.MAXDIST$, always decreases and is set to d_m (step 9). When the highest LOD is reached, the distance range r collapses to an exact value, and the infimum $r.MINDIST$ is also set to d_m (step 11). Meanwhile, the value of *MINMAXDIST* is updated every time that a smaller *MAXDIST* is encountered (step 13).

After checking all the candidate objects with the faces decoded at a certain LOD, some candidates may not qualify anymore and can be removed from the candidate list (steps 14-16). When only one object is left in the *candidates* list, it will be reported as the nearest neighbor of o (step 18). With this method, the nearest neighbor for some objects can be determined before decoding objects into the highest LOD, thus the burden of decoding and geometric computation at high LODs can be mitigated, and the overall query performance is significantly improved as will be demonstrated with our experiments in Section 6. Meanwhile, *k Nearest Neighbor* query (kNN) can be implemented with the Filter-Progressive-Refine paradigm by keeping at least k entries in the *candidates* list.

4.4 Choice of Level of Details

For the progressive refinement conducted for all three types of queries, the LODs that the refinements need to be performed on are recorded in list *lod_list* in ascending order. The list is ended with the highest LOD to make sure accurate results can always be returned. The decision to choose which LODs for refining objects directly impacts overall query performance. On one hand, additional computations are involved in conducting every round of refinement on the polyhedrons at a certain LOD. On the other hand, refining the polyhedrons at a certain LOD may reduce the size of the candidate list thus saving some heavy geometric computations conducted on polyhedrons at higher LODs.

Taking the nearest neighbor query as an example, the number of face pairs evaluated to get the distance between the representations of two objects at LOD i is $N_1^i \times N_2^i$, where N_j^i is the number of faces for the representation of object j at LOD i. When refinement is conducted over n_e object pairs at LOD i and n_p object pairs are pruned, then the amount of computations conducted in refining at LOD i is $n_e \times N_1^i \times N_2^i$, and the amount of computations saved by pruning those n_p object pairs is $n_p \times N_1^{i+1} \times N_2^{i+1}$. To make the refining at LOD i beneficial, $n_e \times N_1^i \times N_2^i$ must be smaller than $n_p \times N_1^{i+1} \times N_2^{i+1}$. Let us assume N_j^{i+1} equals $r \times N_j^i$, then the value of $\frac{n_p}{n_e}$ must be larger than $\frac{1}{r^2}$. Note that simplifying the surface complexity of the polyhedron reduces the face density but preserves the topology of the surface elements. As a result, the portion of vertices removed in each round of decimation is a constant, thus the value of r is a constant. The value of r for typical spatial data and the impact of choosing appropriate LODs on queries will be evaluated with experiments in Section 6.5.

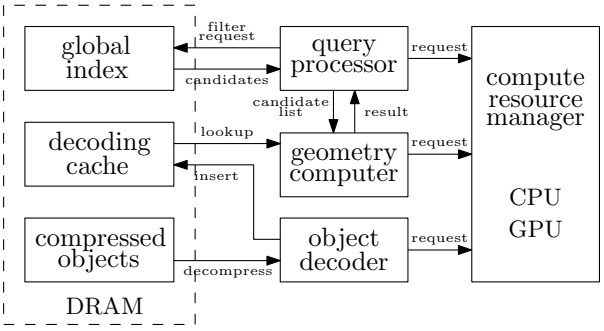


Figure 8: Overview of the architecture of 3DPro

5 IMPLEMENTATION OF 3DPRO

In this section, we provide further implementation details of 3DPro about how other acceleration approaches are implemented along with the Filter-Progressive-Refine paradigm and how the memory and computing resources are managed to facilitate progressive refinement. Fig. 8 illustrates the architecture of 3DPro. After receiving a spatial query request, the query processor searches the global index to retrieve the candidate objects. The candidate objects are then evaluated progressively with the geometry computer starting from lower LODs as the progressive refinement step. The 3D object decoder works on decoding the compressed objects into target LODs.

5.1 Techniques to Reduce Geometric Computation Costs

3DPro equipped the geometry computer with three existing techniques to support efficient geometric computations among individual geometries decoded at each LOD. The FPR paradigm can coexist with all of them to minimize the geometric computations and further boost their efficiency.

Indexing individual objects with AABB-tree: Similar to *R-tree*, an AABB-tree constructs a hierarchical structure of Axis-Aligned Bounding Boxes (AABBs) of the polyhedron primitives [6, 13]. In our implementation, after decoding two objects o and o' to a certain LOD, an AABB-tree is built with the AABBs of all the faces of o', and then the AABB-tree is queried with the AABBs of all the faces of o to accelerate the distance calculation and spatial relationship determining at that LOD. Utilizing AABB-tree reduces the computational complexity of evaluating two geometries from $O(N_l N_l')$ to $O(N_l log(N_l'))$, where N_l and N_l' are the numbers of faces of the polyhedrons representing objects o and o' at LOD l respectively.

Partitioning complex 3D objects: Another technique implemented and compared in 3DPro is the one that partitions the original object into simple sub-objects. Each sub-object is then approximated with Orientable-Bounding Boxes (OBBs) [26] or Minimum Bounding Boxes (MBBs) [6]. Those boxes, instead of a single MBB, are indexed in the global spatial index to achieve a better filtering efficiency. In the filtering step, a subset of sub-objects instead of the entire object is retrieved as the candidate.

In the refinement step, while processing data at a specific LOD, the decoded faces are grouped and assigned to proper candidate boxes for further processing. There exist multiple methods to partitioning 2D or 3D spatial objects [4, 6, 26], of which 3DPro adopts the skeleton-based approach [4]. Initially, a list of skeleton points is extracted from the target geometry, and then each primitive of the geometry representing the object at the highest LOD is assigned to the skeleton point that is **closest** to the primitive. Finally, a group is formed for each skeleton point with the faces assigned to it, and an MBB is generated with those faces.

GPU-based Parallelization: The repetitive tasks for evaluating face pairs fit nicely with GPU parallelism and can be accelerated by general-purpose GPU computing [51]. During processing, a computation buffer is maintained in the memory of each GPU. The face pairs which need to be checked are packed in the buffer. Then a list of computation tasks with the same number of face pairs that need to be evaluated is generated and each task is processed with a single GPU kernel. Operators like intersection detection and distance calculations are implemented as kernel functions. With this method, geometric computations are conducted with GPU in a massively parallel process.

5.2 Computation Resource Management

As demonstrated in Section 6, geometric computation and mesh decoding dominate the execution of all types of spatial join queries. The availability of the computation resources like the GPU and CPU determines the overall processing performance. 3DPro provides transparent management of the CPU cores and GPUs with an independent resource management module. The geometric computations between decoded geometries are grouped into small tasks with a fixed number of face pair evaluations. Those tasks are then completed quickly by CPU cores or GPU when either of them is available, which fully exploits all the computation capability of CPU and GPU. For the purpose of experimental evaluation (Section 6), only CPU or GPU will be used for each test to evaluate the performance of different methods. Besides, the query processor, geometry computer, and object decoder will compete for the CPU resources during execution.

5.3 Memory Management

3DPro takes a memory-centered approach for data management and processing and manages compressed 3D objects and their indexes in memory. The objects are partitioned into fixed-sized cuboids according to their positions. The compressed data for the objects in the same cuboid are stored in the same file for persistent and loaded to a continuous memory region for in-memory computing. For each object, the MBB is read out from the compressed data and assigned with a unique ID and the memory location where its compressed data are stored. An *R-tree* is built with the MBBs for all the objects.

Cache Management for Decoded Data: To avoid repetitive data decoding, which is compute-intensive, 3DPro maintains a Least Recently Used (LRU) decoding cache which keeps early decoded data for reuse. Each entry of the cache is an area of memory that contains the decoded faces for one object at a certain LOD. Thus the handle of the cache entry is composed of the object ID and the LOD. During query processing, whenever the geometry computer fails to find data for a specific object at a certain LOD in the cache, it sends a request to the object decoder and waits for the data to be decoded to such LOD and served in the cache. For all queries conducted on two data sets, data in both sets are partitioned following the same space partition scheme, and the queries referring to the objects in the same cuboids will be conducted in batch. By doing so, recently decoded data have a higher possibility to be reused by serving queries conducted on the objects in the same cuboids thanks to the spatial locality. Objects in multiple cuboids are processed simultaneously for parallelization. The decoding of objects is thread-safe with the protection of cuboid level locks. Note that the objects in different cuboids are indexed in the same global index, thus the objects crossing cuboid boundaries need not be handled separately.

6 EXPERIMENTAL EVALUATION

We compare the efficiency of the Filter-Progressive-Refine paradigm over the traditional Filter-Refine paradigm by conducting all types of 3D spatial queries implemented in Section 4 on 3DPro. Besides, we also run all the tests on PostGIS [49], an open-sourced spatial data management system with 3D support.

6.1 Experimental Setup

All the tests are conducted on a compute node with a 24-core CPU (AMD Ryzen Threadripper 3960X at 3.8GHz) and a GPU (NVIDIA GeForce RTX 2080 Ti with 11GB memory and 4,352 cores). The node comes with 128GB memory (DDR4 3200) and a 2TB SSD (NVMe M.2 PCI-Express 3.0). 80GB memory space is allocated as the decoding cache to host the decoded objects and surface elements. The OS is Ubuntu 18.04.4 with 5.3.0-40 kernel. The CUDA version is 11.4, and the PostGIS version is 3.1.0. We run all tests with multiple threads to exploit all the CPU resources. Before querying, all the data utilized by 3DPro and PostGIS are loaded in memory, thus no I/O is involved in all tests.

6.2 Dataset

We conduct tests over two types of datasets to evaluate the performance of 3DPro and other systems. The first dataset contains about 10 million nuclei which have regular shapes with an average of 300 surface faces per object. The second dataset contains about $50,000$ vessels with bifurcation with a large number of surface faces for each vessel. Each vessel contains an average of $30,000$ faces and 5 bifurcations. The objects in the same dataset do not intersect with each other. The 3D objects are generated by reconstructing 2D segmented objects from adjacent tissue slides from brain tissues [35, 36].

Data partitioning: As is discussed in Section 5.3, data objects are stored in partitions according to their positions. Different from the geospatial data, the objects in the same human tissue have a more balanced distribution, thus both the nuclei and vessels are uniformly distributed in the sampled tissue. All the 3D objects are in a space that is partitioned into $1,000$ cuboids. Each cuboid covers around 50 vessels and $10,000$ nuclei and takes about 20MB space after compression which is a reasonable size for a storage unit. The decoding cache is set to 80GB which is large enough to host the decoded data for all the objects processed in the past 50 partitions and the repeated data decoding is almost eliminated.

Data compression: All objects are represented with polyhedrons at 6 LODs, LOD^0 to LOD^5, where LOD^5 represents the original polyhedron. Details about how the LODs are chosen can be found in Section 6.5. The total size of the fully decoded objects is about 1.15TB and the size of the compressed objects is 18.4GB which can fit into memory comfortably. The compression ratio is high because the PPVP compression applies techniques like spatial compression, entropy encoding, and adaptive quantization introduced in [38]. The vertices and the connection information for the base polyhedron (LOD^0) and the vertices removed during encoding at all the LODs are stored together for the same object

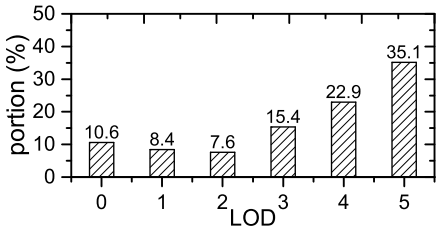


Figure 9: Portions of space taken by different LODs

in one uniformed file format. Fig. 9 shows the portions of the space taken by the base (LOD^0) and the other LODs. Note that decoding one object to a specific LOD also needs to retrieve the data for all the LODs lower than that LOD.

Portion of protruding vertices: As shown in Fig. 4, the shape of a nucleus is almost convex, and as many as 99% of the vertices in the nuclei dataset are protruding. In contrast, as shown in Fig. 1, the shapes of vessels are bifurcated, and there exist many recessing parts like the joint of two branches. As a result, about 75% of the vertices for the vessel dataset are protruding. In total, 92% of the vertices for the polyhedrons representing the data used in our tests in all the LODs are protruding which proves the claim we made in Section 3.2.

Compression cost: On average, it takes 0.4 and 36.3 milliseconds to compress one nucleus and one vessel with the PPVP compression respectively. It takes 120 seconds to compress the data in our experiments with 48 threads. Furthermore, another 308 seconds are used to load the raw data to the CGAL [14] in-memory data structures which is a necessary cost for tests with or without progressive refinement. Note that the compression is a one-time process for each dataset and the cost is negligible. The preprocessing time can also be hidden by the cost of I/O if the uncompressed objects need to be loaded from disks.

6.3 Experimental Design

We evaluate the efficiency of the Filter-Progressive-Refine paradigm on three types of spatial joins based on the queries introduced in Section 4. We tested the intersection spatial join over two nuclei datasets derived from two alternative methods. In digital pathology, it is a common approach to evaluate the effectiveness of one image analysis algorithm by comparing the spatial objects it generates with one ground truth data with an intersection spatial join [62]. We evaluate the efficiency of the FPR paradigm on the nearest neighbor join with two types of dataset combinations: nuclei versus nuclei (nuclei-nuclei) and nuclei versus vessels (nuclei-vessels). That is, given two datasets D_1 and D_2, for each object o in D_1, get the object in D_2 that is closest to it. D_1 is a set of nuclei, and D_2 could be a set of nuclei or vessels. Similarly, the within join is also tested with those two dataset combinations. As is listed in Table 1, there are a total of 5 different types of tests. Each type of test is named X-YZ, where X represents the query types: INT for intersection, WN for within, and NN for nearest neighbor. Y and Z represent the types of the first dataset and the second dataset respectively: N for nuclei and V for vessels. For instance, NN-NV represents conducting nearest neighbor join over a nuclei dataset and a vessel dataset.

To evaluate the efficiency of the existing accelerating approaches, each type of test is conducted in many rounds with different combinations of accelerating approaches applied (partition, AABB, and GPU). A test that conducts geometric computations in a brute-force way is compared as the baseline. Besides, each of those tests is conducted twice with the Filter-Refine (FR) Paradigm and the Filter-Progressive-Refine (FPR) Paradigm to show the efficiency of the novel progressive refinement. The query latencies for all tests are recorded and listed in Table 1. During the execution of each test, the times spent on filtering, decompressing, and conducting geometric computations are profiled respectively for a better understanding of the query processing. Fig. 10 shows the breakdown of the execution time for all tests. Next, we will analyze the results in detail for each query.

6.4 Query Result Analysis

Intersection spatial join: Fig. 10 (INT-NN) shows the time spent by conducting intersection join with different accelerating methods. Note that the evaluating process returns immediately when one pair of intersected faces are detected, thus the intersection detection among two decoded objects can be very efficient, and the amount of geometric computation is much less compared with other distance-based queries. As a result, decompressing objects to higher LODs dominates the execution, and all existing methods compared in the test do not improve the overall query performance significantly. In contrast, when the FPR paradigm is applied (right bars), much fewer objects need to be decoded into higher LODs, thus both the decompression and geometric computations are minimized. Overall, applying the FPR paradigm achieves at least 4X performance improvement even with other accelerating methods applied. It can also be observed that filtering takes a tiny portion of the execution time for all types of queries. It proves that the refinement step dominates the processing cost of 3D spatial queries.

Within spatial join: Fig. 10 (WN-NN) and 10 (WN-NV) show the results of all within spatial join tests with different accelerating approaches over two different dataset combinations using 3DPro with the traditional FR paradigm (left bars) and the FPR paradigm (right bars) respectively. Different from the intersection query, all pairs of faces for two polyhedrons need to be evaluated to determine the distance between them, thus conducting geometric computations will dominate the execution. As those three existing methods majorly improve the performance of conducting the geometric computations, most of those methods can significantly reduce the overall query latency. One exception is the within join over nuclei-nuclei datasets with the partition-based method applied. As the shapes of nuclei are simple, very few of the nuclei are partitioned and the efficiency of partitioning the object into simpler objects is not obvious. In contrast, for the same query over nuclei-vessel datasets, the partition-based method achieves 39X performance improvement over calculating distances in a brute-force way. This is because the shapes of vessels are complex, partitioning them into simpler objects could significantly improve the efficiency of filtering with finer-grained MBBs and reduce the total amount of geometric computations.

It also shows that the FPR paradigm can co-exist with all those methods to further improve the within join query performances for querying both data combinations. For the nuclei-nuclei test (WN-NN), the FPR paradigm achieves 20.9X, 6.4X, and 4.1X performance boosts when partition-based, AABB-tree-based, and GPU-based methods are applied respectively. For the nuclei-vessel test (WN-NV), those performance boosts rates are 5.8X, 5.7X, and 3.3X respectively. Note that, for the nuclei-vessel test, the partition-based method and GPU-based method can co-exist to achieve even better query performance. Therefore, we conducted two more tests with both partition and GPU-based accelerating methods applied. As shown in the last column of

Table 1: The execution time (Seconds) of tests for three queries with different datasets and accelerating approaches

Query	Datasets	Test ID	Paradigm	Brute-force	Partition	AABB	GPU	Partition+GPU
Intersect	Nuclei-Nuclei	INT-NN	FR	356.0	355.7	338.2	340.4	N/A
			FPR	**84.8**	**86.4**	**82.7**	**80.7**	N/A
Within	Nuclei-Nuclei	WN-NN	FR	2253.7	2249.0	480.2	250.8	N/A
			FPR	**108.2**	**108.5**	**74.7**	**60.5**	N/A
	Nuclei-Vessel	WN-NV	FR	25056.8	645.1	11197.3	9827.0	196.3
			FPR	**8458.8**	**111.6**	**1948.7**	**2990.1**	**95.1**
Nearest Neighbor	Nuclei-Nuclei	NN-NN	FR	2264.0	2268.9	516.9	267.9	N/A
			FPR	**893.8**	**893.1**	**306.6**	**164.1**	N/A
	Nuclei-Vessel	NN-NV	FR	151630.0	1649.8	108799.9	62506.1	392.8
			FPR	**24968.1**	**422.2**	**21025.6**	**10202.0**	**172.3**

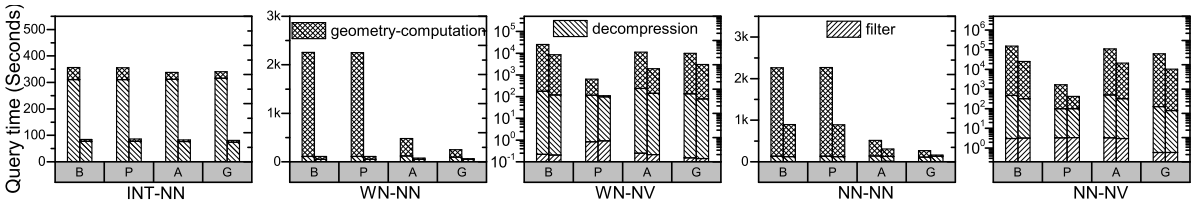


Figure 10: Performance comparison of 3DPro with traditional FR paradigm (left bars) and the proposed FPR paradigm (right bars) utilizing different accelerating approaches: Brute force(B), Partition(P), AABB(A), and GPU(G) on all tests

Table 1, using GPU can further reduce the query latency by 70% (645.1 to 196.3) when the vessels are already partitioned into simple objects. As can be expected, the FPR paradigm can further improve the query performance by 2 folds (196.3 to 95.1).

Nearest neighbor join: Fig. 10 (NN-NN) and 10 (NN-NV) show the results for all the nearest neighbor join tests. It can be observed that the execution time of the nearest neighbor join query over different datasets with different accelerating methods follows a similar pattern as the within join query, and conducting geometric computations dominates the execution. The AABB and GPU-based methods work for both data combinations and the partition-based method only works for the test with the nuclei-vessel datasets. When the FPR paradigm is applied, the nearest neighbor join query can be further accelerated for all tests. For the nuclei-nuclei test (NN-NN), the FPR paradigm further improves the query performance by 2.5X, 1.7X, and 1.6X when partition, AABB-tree, GPU based methods are applied respectively. For the nuclei-vessel test (NN-NV), the FPR paradigm boosts the query efficiency by 3.9X, 5.2X, and 6.1X when partition, AABB-tree, GPU based methods are applied respectively. When both the partition and GPU-based methods are applied, the FPR paradigm further reduces the execution time by 56% (392.8 to 172.3).

Table 2: The total decoding time (in Seconds) for tests with and without the decoding cache.

Test ID	without cache	with cache	speedup
INT-NN	325.7	309.7	1.1
WN-NN	496.7	110.8	4.5
WN-NV	11134.9	175.4	63.5
NN-NN	300.4	125.9	2.4
NN-NV	106906.8	463.8	230.5

Efficiency of the decoding cache: 3DPro maintains an LRU decoding cache to keep the faces for the recently decoded objects. Table 2 lists the decoding time taken by different tests with and without the decoding cache enabled. It shows that the decoding time is significantly reduced when the decoding cache is enabled and the recently decoded objects are cached for future reuse. It also shows that the decoding time reduction is even more significant when vessels are involved. That is because one vessel might be the within or nearest neighbor candidate for hundreds of nuclei, such that each vessel needs to be decoded hundreds of times without the help of the decoding cache. Note that the decoding cache is enabled by default for all tests.

Repeated face pair evaluation: Note that one face may be shared by the polyhedrons in multiple LODs for the same spatial object, thus the same pair of faces may be repeatedly evaluated while conducting refinement at two LODs. In practice, the majority of the faces in low LOD polyhedrons will be replaced in the process of decoding to a higher LOD. By profiling the compressed data, we find that an average of about 15.6% of the faces are shared by at least two LODs of polyhedrons. In addition, repeated face pair evaluation happens only when both the evaluated faces are shared by the same two LODs of polyhedrons. As a result, the potion of the repeatedly evaluated face pairs, or the redundant evaluations, is much less, which is only about 1.8% in our test.

Overall analysis: Conclusions can be drawn based on the experiments described above. First, the AABB-tree-based method, which indexes the faces of the decoded polyhedron with a hierarchical tree structure, can reduce the computation complexity for all the tests but the efficiency is not as good as the other methods for most cases. Second, the GPU-based method can also help to conduct the geometric computations much faster than the multi-core CPU for all the tests. Third, the partition-based method can significantly reduce the total amount of geometric computations involved in the refinement step only when the queried polyhedrons are complex in shape but does not help much when the objects are simple. Fourth, the FPR paradigm reduces the complexity of the objects involved in the refinement step which further reduces the amount of geometric computations from another

dimension. As a result, it can work together with any other existing accelerating method to achieve optimized 3D spatial queries. Fifth, decompressing objects is time-consuming and even dominates the execution in some of the tests, such as the intersection tests (Fig. 10 INT-NN) and within query on nuclei-nuclei data set when the FPR paradigm is applied (Fig. 10 WN-NN right bars). The decompression efficiency of 3DPro is highly bounded by the performance of a third-party dependency library CGAL [14]. We will decouple those dependencies and implement the spatial decompression on GPU as future work.

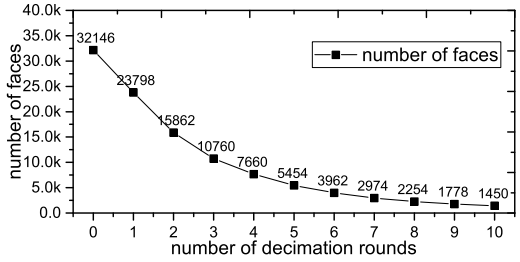


Figure 11: The number of remaining faces vs the number of decimation rounds

6.5 Choosing Level of Details with Profiling

As is discussed in Section 4.4, one critical parameter that affects the performance of 3DPro with the FPR paradigm is at which LODs representation objects need to be refined during the progressive refinement step. To determine on which LOD the objects need to be refined, we conduct all the queries mentioned above on the objects in a single cuboid which contains 45 vessels and 9, 945 nuclei. All the join tests are implemented with 9, 945 independent queries over individual objects, and for each independent query, all LODs are checked to progressively prune the candidates until the final result is determined. Each query needs to evaluate multiple pairs of objects in the refinement step after filtering with the index, and at each LOD, all the object pairs need to be evaluated but only some of them will be pruned.

To avoid too many faces being shared by two consecutive LODs, we denote that the LOD is reduced by one level after two rounds of decimation. As shown in Fig. 11, the number of faces decreases half after two rounds of decimation, thus r equals 2 for all LODs. Furthermore, a polyhedron representing a nucleus contains only about 10 faces after 10 rounds of decimation, which cannot be further compressed. To make it consistent, 6 levels of LODs (1 original and 5 compressed) for both nuclei and vessels datasets are chosen. As the value of r equals 2, the value of $\frac{n_p}{n_e}$ in Section 4.4 must be larger than $\frac{1}{4}$ or 25%. In other words, at least a quarter of the object pairs should be pruned to make the refinement at a specific LOD beneficial. Fig. 12 shows the total number of object pairs that are evaluated/pruned and the portion of the pruned object pairs at each LOD for all types of queries. The LOD needs to be refined when the portion is larger than 25%. For instance, the INT-NN test needs to conduct progressive refinement on LODs 0, 1, and 5. Note that when the partition-based approach is applied, the vessel dataset can be treated as a different dataset with simpler objects. As the portion of the pruned object pairs at each LOD is highly dataset depended, for each test, one round of profiling can be conducted automatically with the sampled objects to determine the LODs the progressive refinements need to be conducted on.

6.6 Comparison with PostGIS

We further compare the performance of 3DPro with PostGIS in supporting all those three types of spatial queries over 3D data. As 3D join queries over complex 3D objects is to be optimized in PostGIS, it may take too long to complete the queries [22, 51]. For convenience, we conducted the join queries only on one cubic of the data which contains 45 vessels and 9945 nuclei. We compare the average latency of those queries with the average latency of tests using 3DPro with the FR or the FPR paradigm. For fairness, the geometric computations in 3DPro are conducted in a brute-force way without the help of any other facilitating approaches like AABB-tree and GPU. Each test is conducted with a single thread for both 3DPro and PostGIS. All the data accessed is in memory thus no I/O is involved for both PostGIS and 3DPro. Furthermore, as PostGIS does not support nearest neighbor candidates filtering with indexing [48], for fairness, we calculate the distances from all geometries to their nearest neighbors with 3DPro and use the largest distance as a buffer size. To get the nearest neighbor of a geometry with PostGIS, a buffer geometry is created with the buffer size, and all the geometries in the source dataset with MBB intersecting the buffer geometry are retrieved as nearest neighbor candidates.

Fig. 13 shows the query latency of all queries using PostGIS and using 3DPro with FR or FPR paradigm. It shows that PostGIS is up to two orders of magnitude slower than 3DPro even with the Filter-Refine paradigm. In contrast, when the Filter-Progressive-Refine paradigm is applied, the query can be further accelerated by up to one order of magnitude. This set of tests proves that the state-of-the-art spatial data management systems (SDBMS) are not optimized for supporting queries over complex 3D spatial objects. All existing accelerating methods, together with the FPR paradigm proposed in this paper, can be applied to improve the query performance of the state-of-the-art SDBMS by up to 5 orders of magnitude for certain types of queries.

7 RELATED WORK

3D spatial data processing: Support of 3D objects with a comprehensive 3D model is limited in traditional spatial database management systems. PostGIS [49] recently introduced 3D support using a surface-based model, where no data compression, multiple levels of details, or 3D specialized indexing is adopted, leading to limited performance for complex 3D objects. LumicGIS boosts 3D spatial queries on top of PostGIS with the help of GPU [51], but it does not reduce the total load of geometric computations. Oracle Spatial takes a method on building an *R-tree* on the primitives of complex geometries to facilitate queries [41, 50]. However, none of them takes advantage of multiple LODs, and the Filter-Progressive-Refine paradigm can coexist with these systems for further efficiency improvement.

Spatial compression in other domains: Spatial compression is an important research topic in computational geometry and computer graphics. One research area focuses on minimizing the distortion of the compressed object. Peng et. al proposed an OCTree-based method to improve the efficiency of encoding meshes with arbitrary topological structures [47]. They also proposed a feature-oriented generic progressive lossless mesh coder which focuses more on the presentation of the compressed objects [46]. Yet another Incremental Parametric Refinement based approach is proposed which also improves the quality of compression using a novel refinement scheme [59]. PPMC achieves a lower distortion rate with a lifting schema and improves the

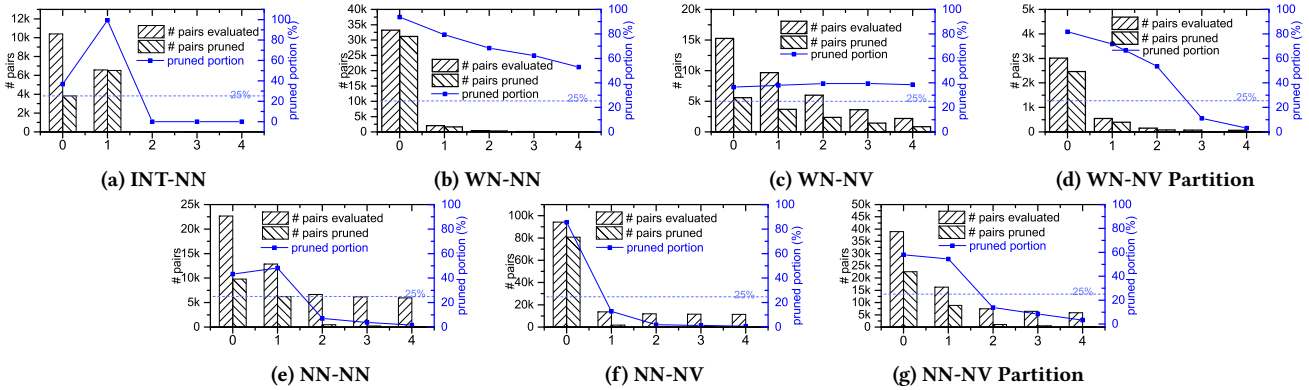


Figure 12: The number of object pairs evaluated and pruned with refinements performed at different LODs

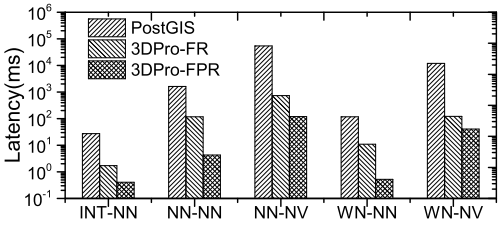


Figure 13: The latency of tests querying with PostGIS and 3DPro with FR/FPR paradigms

compression quality [38]. Furthermore, multiple LOD representations are first introduced in the computer graphics field for faster rendering and visualization [37]. For instance, [18] proposed a hierarchical representation for efficient graph rendering with GPU. In contrast, the progressive refinement method proposed in this paper requires that the compressed objects should preserve a spatial relationship with the original object (covering a subset) to make querying with the compressed objects possible.

Wavelet-based compression: Wavelet implements multiple resolution representations of 3D objects with slightly different underlying logic [55]. With wavelet representation, the original polyhedron is re-meshed to make the edges follow a certain pattern with a sacrifice of precision, then the re-meshed polyhedron is compressed with wavelet transforms. The wavelet-based compression is then implemented over *normal meshes* representations to improve the compression efficiency [31]. Instead of selecting the removed vertices arbitrarily, the selected vertices and edges in wavelet representation follow a certain topology structure, thus progressive rendering and querying can be achieved by fetching the details of an object in batches. [45] studied methods to improve the storage efficiency of wavelet-based compression. [2, 3] proposed a motion-aware approach that retrieves objects in proper resolution considering the movement of the objects. The motion-aware approach significantly reduced the I/O and computation cost with the dynamical loading. We will adapt the Filter-Progressive-Refine paradigm to all those progressive compression methods as future work.

Researches in spatial join: Spatial join is well studied in both 2D and 3D spaces [8–11, 40, 66]. However, most of the objectives are to minimize the I/O cost and implement spatial joins over a large chunk of data with limited memory capacity [9–11]. TOUCH achieves efficient in-memory spatial join by using hierarchical data-oriented space partitioning [40]. A GPU-based approach is proposed in [66] which rasterizes polygons into labeled grids and conducts spatial join over those grids. [24] proposes scalable algorithms for nearest neighbor join for trajectory data. Besides Minimum Bounding Boxes (MBBs), [9, 10] study the efficiencies of many other approximation methods like Maximum Enclosed Boxes, Minimum Bounding Ellipse, and Convex Hulls in filtering out more objects. [53] proposes Clipped Bounding Box (CBB) for a better approximation of the region covered by spatial index nodes. [57, 67] takes advantage of raster representations of objects to achieve a better approximation. They all aim to improve the efficiency of the filtering step, not on easing geometric computations introduced in the refinement step.

Distributed spatial data management: There is much recent work on scalable 2D spatial data management with distributed data processing frameworks, such as Hadoop-GIS [1], SpatialHadoop [21], GeoSpark [65], Simba [64], SparkGIS [5], and LocationSpark [56]. CG_Hadoop implements common computational geometry operations with Hadoop framework [20]. iSPEED is a recent research project on supporting 3D data in distributed and in-memory computing, intending to achieve fast query response by sacrificing the accuracy of queries using low-resolution objects from progressive compression [35, 61]. Those distributed big spatial data processing systems can benefit from the progressive refinement proposed in this work.

8 CONCLUSION

In this paper, we propose a new Filter-Progressive-Refine paradigm for 3D spatial queries at an extreme scale. 3DPro aims to provide high-performance 3D spatial queries with effective access methods, in-memory computing, complexity reduction, and CPU-GPU hybrid parallelization. The progressive protruding-vertex pruning-based compression not only significantly reduces the data size for in-memory processing, but also provides hierarchical encoding to progressively reduce the surface complexity. With its unique properties preserved during encoding, the progressive compression enables progressive query processing to minimize decoding and geometric computation. 3DPro achieves significant performance improvement over state-of-the-art 3D spatial data management systems and provides a promising approach for many 3D applications.

ACKNOWLEDGMENTS

This research is supported in part by grants from National Science Foundation ACI 1443054 and IIS 1541063, and National Institute of Health U01CA242936.

REFERENCES

[1] Ablimit Aji, Fusheng Wang, Hoang Vo, Rubao Lee, Qiaoling Liu, Xiaodong Zhang, and Joel Saltz. 2013. Hadoop gis: a high performance spatial data warehousing system over mapreduce. *Proceedings of the VLDB Endowment* 6, 11 (2013), 1009–1020.

[2] Mohammed Eunus Ali, Egemen Tanin, Rui Zhang, and Lars Kulik. 2010. A motion-aware approach for efficient evaluation of continuous queries on 3d object databases. *The VLDB journal* 19, 5 (2010), 603–632.

[3] Mohammed Eunus Ali, Rui Zhang, Egemen Tanin, and Lars Kulik. 2008. A motion-aware approach to continuous retrieval of 3d objects. In *2008 IEEE 24th International Conference on Data Engineering*. IEEE, 843–852.

[4] Oscar Kin-Chung Au, Chiew-Lan Tai, Hung-Kuo Chu, Daniel Cohen-Or, and Tong-Yee Lee. 2008. Skeleton extraction by mesh contraction. *ACM transactions on graphics (TOG)* 27, 3 (2008), 1–10.

[5] Furqan Baig, Hoang Vo, Tahsin Kurc, Joel Saltz, and Fusheng Wang. 2017. Sparkgis: Resource aware efficient in-memory spatial query processing. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 28.

[6] Gino van den Bergen. 1997. Efficient collision detection of complex deformable models using AABB trees. *Journal of graphics tools* 2, 4 (1997), 1–13.

[7] Christian Böhm, Stefan Berchtold, and Daniel A Keim. 2001. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys (CSUR)* 33 (2001), 322–373.

[8] Panagiotis Bouros and Nikos Mamoulis. 2019. Spatial joins: what's next? *SIGSPATIAL Special* 11, 1 (2019), 13–21.

[9] Thomas Brinkhoff, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1994. Multi-step processing of spatial joins. *Acm Sigmod Record* 23, 2 (1994), 197–208.

[10] Thomas Brinkhoff, Hans-Peter Kriegel, and Bernhard Seeger. 1993. Efficient processing of spatial joins using R-trees. *ACM SIGMOD Record* 22, 2 (1993), 237–246.

[11] Thomas Brinkhoff, H-P Kriegel, and Bernhard Seeger. 1996. Parallel processing of spatial joins using R-trees. In *Proceedings of the Twelfth International Conference on Data Engineering*. IEEE, 258–265.

[12] cesium. accessed June, 2021. *cesium*. https://cesium.com/

[13] CGAL. accessed June, 2021. *AABB-tree*. https://doc.cgal.org/latest/AABB_tree/index.html

[14] CGAL. accessed June, 2021. *The Computational Geometry Algorithms Library*. http://www.sfcgal.org/

[15] King Lum Cheung and Ada Wai-Chee Fu. 1998. Enhanced nearest neighbour search on the R-tree. *ACM SIGMOD Record* 27, 3 (1998), 16–21.

[16] HuBMAP Consortium et al. 2019. The human body at cellular resolution: the NIH Human Biomolecular Atlas Program. *Nature* 574, 7777 (2019), 187.

[17] L. A. D. Cooper, A. B. Carter, A. B. Farris, F. Wang, J. Kong, D. A. Gutman, P. Widener, T. C. Pan, S. R. Cholleti, A. Sharma, T. M. Kurc, D. J. Brat, and J. H. Saltz. 2012. Digital Pathology: Data-Intensive Frontier in Medical Imaging. *Proc. IEEE* 100, 4 (April 2012), 991–1003.

[18] Mark Dokter, Jozef Hladky, Mathias Parger, Dieter Schmalstieg, Hans-Peter Seidel, and Markus Steinberger. 2019. Hierarchical Rasterization of Curved Primitives for Vector Graphics Rendering on the GPU. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 93–103.

[19] David H Douglas and Thomas K Peucker. 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization* 10, 2 (1973), 112–122.

[20] Ahmed Eldawy, Yuan Li, Mohamed F Mokbel, and Ravi Janardan. 2013. CG_Hadoop: computational geometry in MapReduce. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 294–303.

[21] 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.

[22] Esri. accessed June, 2021. *3D GIS*. https://www.esri.com/en-us/arcgis/3d-gis/

[23] Orit Rozenblatt-Rosen et al. 2020. The Human Tumor Atlas Network: Charting Tumor Transitions across Space and Time at Single-Cell Resolution. *Cell* 181, 2 (2020), 236 – 249. https://doi.org/10.1016/j.cell.2020.03.053

[24] Yixiang Fang, Reynold Cheng, Wenbin Tang, Silviu Maniu, and Xuan Yang. 2015. Scalable algorithms for nearest-neighbor joins on big trajectory data. *IEEE Transactions on Knowledge and Data Engineering* 28, 3 (2015), 785–800.

[25] Navid Farahani, Alex Braun, Dylan Jutt, Todd Huffman, Nick Reder, Zheng Liu, Yukako Yagi, and Liron Pantanowitz. 2017. Three-dimensional imaging and scanning: current and future applications for pathology. *Journal of pathology informatics* 8 (2017).

[26] Stefan Gottschalk, Ming C Lin, and Dinesh Manocha. 1996. OBBTree: A hierarchical structure for rapid interference detection. In *Proceedings of the 23rd ACM SIGGRAPH annual conference on Computer graphics and interactive techniques*. 171–180.

[27] Antonin Guttman. 1984. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*. 47–57.

[28] Gísli R Hjaltason and Hanan Samet. 1999. Distance browsing in spatial databases. *ACM Transactions on Database Systems (TODS)* 24, 2 (1999), 265–318.

[29] Hugues Hoppe. 1996. Progressive meshes. In *Proceedings of the 23rd ACM SIGGRAPH annual conference on Computer graphics and interactive techniques*. 99–108.

[30] 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.

[31] Andrei Khodakovsky and Igor Guskov. 2004. Compression of normal meshes. In *Geometric modeling for scientific visualization*. Springer, 189–206.

[32] Andrei Khodakovsky, Peter Schröder, and Wim Sweldens. 2000. Progressive geometry compression. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 271–278.

[33] Jun Kong, Fusheng Wang, George Teodoro, Yanhui Liang, Yangyang Zhu, Carol Tucker-Burden, and Daniel J Brat. 2015. Automated cell segmentation with 3D fluorescence microscopy images. In *2015 IEEE 12th International Symposium on Biomedical Imaging (ISBI)*. IEEE, 1212–1215.

[34] Eric Larsen. accessed June, 2021. *PQP*. https://github.com/GammaUNC/PQP

[35] Yanhui Liang, Hoang Vo, Jun Kong, and Fusheng Wang. 2017. ispeed: an efficient in-memory based spatial query system for large-scale 3d data with complex structures. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 1–10.

[36] Yanhui Liang, Fusheng Wang, Pengyue Zhang, Joel H Saltz, Daniel J Brat, and Jun Kong. 2017. Development of a framework for large scale three-dimensional pathology and biomarker imaging and spatial analytics. *AMIA Summits on Translational Science Proceedings* 2017 (2017), 75.

[37] David Luebke, Martin Reddy, Jonathan D Cohen, Amitabh Varshney, Benjamin Watson, and Robert Huebner. 2003. *Level of detail for 3D graphics*. Morgan Kaufmann.

[38] Adrien Maglo, CléMent Courbet, Pierre Alliez, and Céline Hudelot. 2012. Progressive compression of manifold polygon meshes. *Computers & Graphics* 36, 5 (2012), 349–359.

[39] MIT. accessed June, 2021. *3D Scan*. https://alum.mit.edu/slice/robotic-microscope-brings-pathology-speed

[40] Sadegh Nobari, Farhan Tauheed, Thomas Heinis, Panagiotis Karras, Stéphane Bressan, and Anastasia Ailamaki. 2013. TOUCH: in-memory spatial join by hierarchical data-oriented partitioning. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. 701–712.

[41] Oracle. accessed June, 2021. *Oracle 3D*. https://docs.oracle.com/database/121/SPATL/three-dimensional-spatial-objects.htm#SPATL468

[42] Joseph O'rourke et al. 1998. *Computational geometry in C*. Cambridge university press.

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

[44] Apostolos Papadopoulos and Yannis Manolopoulos. 1997. Performance of nearest neighbor queries in R-trees. In *International Conference on Database Theory*. Springer, 394–408.

[45] Frédéric Payan and Marc Antonini. 2002. 3D mesh wavelet coding using efficient model-based bit allocation. In *Proceedings. First International Symposium on 3D Data Processing Visualization and Transmission*. IEEE, 391–394.

[46] Jingliang Peng, Yan Huang, C-C Jay Kuo, Ilya Eckstein, and M Gopi. 2010. Feature oriented progressive lossless mesh coding. In *Computer Graphics Forum*, Vol. 29. Wiley Online Library, 2029–2038.

[47] Jingliang Peng and C-C Jay Kuo. 2005. Geometry-guided progressive lossless 3D mesh coding with octree (OT) decomposition. In *ACM Transactions on Graphics (TOG)*, Vol. 24. ACM, 609–616.

[48] PostGIS. accessed June, 2021. *Nearest-Neighbour Searching*. https://postgis.net/workshops/postgis-intro/knn.html

[49] Postgres. accessed June, 2021. *PostGIS*. https://postgis.net/

[50] Siva Ravada, Baris M Kazar, and Ravi Kothuri. 2009. Query processing in 3d spatial databases: Experiences with oracle spatial 11g. In *3D Geo-Information Sciences*. Springer, 153–173.

[51] Lucas C Villa Real, Bruno Silva, Dikran S Meliksetian, and Kaique Sacchi. 2019. Large-scale 3D geospatial processing made possible. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 199–208.

[52] Philip Schneider and David H Eberly. 2002. *Geometric tools for computer graphics*. Elsevier.

[53] Darius Sidlauskas, Sean Chester, Eleni Tzirita Zacharatou, and Anastasia Ailamaki. 2018. Improving spatial data processing by clipping minimum bounding boxes. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 425–436.

[54] Leslie Solorzano, Gabriela M Almeida, Bárbara Mesquita, Diana Martins, Carla Oliveira, and Carolina Wählby. 2018. Whole slide image registration for the study of tumor heterogeneity. In *Computational Pathology and Ophthalmic Medical Image Analysis*. Springer, 95–102.

[55] Eric J Stollnitz, Tony D DeRose, Anthony D DeRose, and David H Salesin. 1996. *Wavelets for computer graphics: theory and applications*. Morgan Kaufmann.

[56] Mingjie Tang, Yongyang Yu, Qutaibah M Malluhi, Mourad Ouzzani, and Walid G Aref. 2016. Locationspark: A distributed in-memory data management system for big spatial data. *Proceedings of the VLDB Endowment* 9, 13 (2016), 1565–1568.

[57] Dejun Teng, Furqan Baig, Qiheng Sun, Jun Kong, and Fusheng Wang. 2021. IDEAL: a Vector-Raster Hybrid Model for Efficient Spatial Queries over Complex Polygons. In *2021 22nd IEEE International Conference on Mobile Data*

Management (MDM). IEEE, 99–108.

[58] Oren Tropp, Ayellet Tal, and Ilan Shimshoni. 2006. A fast triangle to triangle intersection test for collision detection. *Computer Animation and Virtual Worlds* 17, 5 (2006), 527–535.

[59] Sébastien Valette, Raphaëlle Chaine, and Rémy Prost. 2009. Progressive lossless mesh compression via incremental parametric refinement. In *Computer Graphics Forum*, Vol. 28. Wiley Online Library, 1301–1310.

[60] Mahes Visvalingam and J Duncan Whyatt. 1990. The Douglas-Peucker algorithm for line simplification: re-evaluation through visualization. In *Computer Graphics Forum*, Vol. 9. Wiley Online Library, 213–225.

[61] Hoang Vo, Yanhui Liang, Jun Kong, and Fusheng Wang. 2018. iSPEED: a scalable and distributed in-memory based spatial query system for large and structurally complex 3D data. *Proceedings of the VLDB Endowment* 11, 12 (2018).

[62] Fusheng Wang, Jun Kong, Jingjing Gao, Lee AD Cooper, Tahsin Kurc, Zhengwen Zhou, David Adler, Cristobal Vergara-Niedermayr, Bryan Katigbak, Daniel J Brat, et al. 2013. A high-performance spatial database based approach for pathology imaging algorithm evaluation. *Journal of pathology informatics* 4 (2013).

[63] Kaibo Wang, Yin Huai, Rubao Lee, Fusheng Wang, Xiaodong Zhang, and Joel H Saltz. 2012. Accelerating pathology image data cross-comparison on CPU-GPU hybrid systems. In *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, Vol. 5. NIH Public Access, 1543.

[64] Dong Xie, Feifei Li, Bin Yao, Gefei Li, Liang Zhou, and Minyi Guo. 2016. Simba: Efficient In-Memory Spatial Analytics. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 1071–1085. https://doi.org/10.1145/2882903.2915237

[65] Jia Yu, Zongsi Zhang, and Mohamed Sarwat. 2019. Spatial data management in apache spark: The geospark perspective and beyond. *Geoinformatica* 23, 1 (2019), 37–78.

[66] Eleni Tzirita Zacharatou, Harish Doraiswamy, Anastasia Ailamaki, Cláudio T Silva, and Juliana Freiref. 2017. GPU rasterization for real-time spatial aggregation over arbitrary polygons. *Proceedings of the VLDB Endowment* 11, 3 (2017), 352–365.

[67] Eleni Tzirita Zacharatou, Andreas Kipf, Ibrahim Sabek, Varun Pandey, Harish Doraiswamy, and Volker Markl. 2021. The Case for Distance-Bounded Spatial Approximations. In *11th Conference on Innovative Data Systems Research, CIDR 2021, Virtual Event, January 11-15, 2021, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2021/papers/cidr2021_paper19.pdf

[68] Andi Zang, Shiyu Luo, Xin Chen, and Goce Trajcevski. 2019. Real-Time Applications Using High Resolution 3D Objects in High Definition Maps (Systems Paper). In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 229–238.

[69] Jun Zhang, Nikos Mamoulis, Dimitris Papadias, and Yufei Tao. 2004. All-nearest-neighbors queries in spatial databases. In *Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004.* IEEE, 297–306.