EVALUATING SPATIOTEMPORAL SEARCH STRUCTURES FOR ANALYSIS FROM LAGRANGIAN BASIS FLOWS

by:

Ouermi Timbwaoaga Aime Judicael (TAJO)

Presented to the Department of Computer and Information Science

at the University of Oregon

in partial fulfillment for the degree of

Bachelor of Science

June 2016

THESIS APROVAL PAGE

Student: Ouermi Timbwaoga Aime Judicael (TAJO)

Title: Evaluating Spatiotemporal Search Structures for Analysis from Lagrangian Basis Flows.

This thesis has been accepted and approved in partial fulfillment of the requirements for the Bachelor of Science degree in the Department of Computer and Information Science by:

Hank Childs	Advisor

Original approval signatures are on file with the Department of Computer and Information Science at the University of Oregon

Degree awarded June 2016

THESIS ABSTRACT

Ouermi Timbwaoga Aime Judicael (TAJO)

Bachelor of Science

Computer and Information Science

June 2016

Title: Evaluating Spatiotemporal Search Structures for Analysis from Lagrangian Basis Flows.

Visualization Approved: ___________________________
Hank Childs

While flow visualization has traditionally been performed from the Eulerian perspective, the Lagrangian approach is gaining momentum in the scientific community. This is because the Lagrangian approach offers more opportunities to mitigate I/O limitations than the traditional Eulerian approach. Particle trajectory tracing using the Lagrangian approach is performed by extracting Lagrangian basis flows, and using these flows to construct new trajectories from particle seed locations. Tracing the trajectory of a given particle requires finding its neighboring basis flows. Our work investigates different spatiotemporal search structures for the purpose of particle trajectory tracing from Lagrangian basis flows. We conducted our study by evaluating the storage size, the build time, and the search time of the different search structures over various configurations.

Key Words:

Lagrangian basis flows, Bounding Volume Hierarchy, K-d Tree.

Table of Contents

<u>List of Tables</u>

1 INTRODUCTION

Flow visualization enables understanding and exploration of fluids, mixing, wind, and many other phenomena. It gives scientists the opportunity to explore phenomena they would not otherwise be able to explore, and plays a vital role in scientific advancement and discoveries in fields such as medicine, mechanical engineering, and astrophysics.

McLouglin et al. [1]surveyed a variety of methods for particle path tracing used in flow visualization. The majority of these flow analysis techniques study the trajectory of particles placed at seed positions. That is, particles are placed at an initial position, and the underlying vector field from the input data set is used to calculate the path each particle follows. A variety of techniques, such as streamlines, line integral convolution [6], pathlines [10], stream surfaces [8], streamlines [11], and FTLE [7], then use these trajectories to form their respective visualizations or analyses.

The type of flow field utilized in flow studies can be categorized in two main groups: “steady” and “unsteady.” A “steady” flow is a flow in which the vector field is independent of time --- the flow does not change as time evolves. On the contrary, “unsteady” flow represents a flow in which the vector field is time dependent. This means that the flow field changes as time progresses.

The traditional method for calculating a particle trajectory begins by evaluating its position with respect to the input data’s vector field, and then displacing it in a small distance along the velocity direction at the particle’s location. This process continues iteratively, with the particle moving small distances each iteration, again in the direction of the vector field at the particle’s current location. Thus, this can be viewed as an ordinary differential equation problem. In order to ensure accuracy in solving the equation, numerical methods such as the Runge-Kutta scheme [9], can be used. In total, calculating the trajectory of a particle can be very computationally intensive because it requires calculating thousands of steps for millions or even billions of particles.

Despite widespread usage, the traditional method for particle trajectories of unsteady flow presents some limitations. The limitations from this method stem from limitations in I/O. First, it is important to note that the simulation calculates the velocity field over all time, but can only save a subset of this information to disk. The most common tradeoff made by simulation codes is to save complete state information for some snapshots in time, and to save no information for the times in between the snapshots. This temporal sparsity introduces errors in interpolation. When tracing the particles' trajectories at the "post hoc" stage, the missing information about the flow between "time slices" may have relevant information about the flow's behavior. The probability of getting inaccurate interpolation increases as the data become sparser. Furthermore, this problem is only getting worse. Despite the fact that I/O bandwidth is increasing in almost every new supercomputer, I/O is not keeping up with their ability to generate data. This leads to more temporal sparsity in the simulation's data saved to disk for post hoc analysis, thus increasing error further.

Fluid mechanics considers two approaches for studying flow: Eulerian and Lagrangian. In the Eulerian approach, a fixed frame of reference is defined in which an observer can watch the flow go by. Traditionally, this framework is what has been used to study flow fields; particles are seeded at different positions and displaced tangent to the velocity of the input data's vector field by solving an ordinary differential equation:

$$\frac{dP}{dt} = V(P, t) \tag{1}$$

Where P(x, y, z, t) defines the particle position.

With sparse data, this introduces errors. The first most significant work on the Lagrangian approach was introduced by G. Haller et al. [12,13] The Lagrangian approach for particle trajectory tracing was introduced as an alternative method in order to resolve the limits with the traditional approach. In this approach, a frame of reference is defined according to the particle's motion, also known as a Lagrangian frame of reference. In this frame, the observer sees the flow from the perspective of the particle. Particles are seeded at different positions and use the neighboring basis flows, which were precomputed, to displace each particle to its next position. This new approach is gaining momentum in the scientific visualization community because it yields better performance and results, and helps mitigate the limits of the previous approaches.

Both paradigms present significant challenges that affect performance and accuracy for flow analysis. The temporal sparsity problem due to the limitation in I/O can be resolved by performing tasks "in situ" [24]. However, this approach implies knowing the type of work to be performed a priori. Thus, the "in situ" paradigm is not well suited for exploration-oriented visualization [29]. In exploration oriented-visualization, the tasks to be performed are not known beforehand because the objective is to investigate the flows interactively to gain insight and learn from them.

Another approach adopted by the scientific community, which helps resolve the sparsity and accuracy issues, is Lagrangian path tracing. In the Lagrangian technique, "time intervals" are used as opposed to "time slices." This minimizes the sparsity in the data and provides better accuracy when interpolation is performed. The Lagrangian method is executed in two phases: "in situ" and "post hoc." The "in situ" phase focuses on extracting the optimal Lagrangian flows. In the "post hoc" phase, seeded particles' paths are traced using the underlying basis flow extracted during the “in situ” stage from a given flow field. Tracing a given particle trajectory requires access to its neighbors to perform an interpolation. The search for the neighbors is computationally intensive, and to our knowledge, no extensive work has been done to evaluate how to optimally locate the neighbors in a spatiotemporal data set of Lagrangian basis flows for visualization.

As mentioned previously, flow anlyses are often very computationally intensive. For a given particle, performing the interpolation, requires finding its neighboring Lagrangian basis flows, and calculating their trajectories. These trajectories are then used to determine the next position of the particle. The nature of the challenges faced in this problem necessitates techniques to diminish the computational burden while tracing the particles’ trajectories.

This study explores new avenues for spatiotemporal search structures for analysis from Lagrangian basis flows. As mentioned earlier, to our knowledge, no ample studies have been conducted to evaluate and determine well-suited search structures for the Lagrangian base flow visualization. Chandler et. al [3] used a modified 3D K-d Tree as an accelerated search structure to find the neighbors of a given particle in the process of tracing its trajectory. Here, we propose a different approach for locating the neighbors of a given particle. We investigated the use of spatiotemporal K-d Trees as search structures. In addition to the K-d Trees, we explore the use of spatiotemporal Bounding Volume Hierarchies (BVH) to evaluate the choice of search structure for particle trajectory tracing purposes. BVH [21] and K-d Trees [15] are often used in ray tracing as an accelerated structure for rendering.

In terms of contribution, this study considers a novel approach for search structure to accelerate Lagrangian particle trajectory tracing, given a spatiotemporal data set. Using spatiotemporal search structures enables us to perform a one-time build instead of building the search structure at each time step. Our research indicates that BVH yields better performance than the K-d Tree approach. This contributes to improving particle trajectory construction during the “post hoc” stage and obtaining significant improvement in performance overall.

Our investigation consisted of three major phases. In the first phase, we built our data structures. As noted before, we considered two versions of 4D K-d Trees and two versions of 4D BVHs. In the second phase we used the already built data structures to perform a spatiotemporal search for neighbors. In the third phase, we analyzed and compared each data structure performance by measuring its build time and search time for a given data sets and seeded particles.

2 RELATED WORK

2.1 Data Compression

Vector field compression is one approach for addressing I/O limitations with traditional Eulerian particle trajectory tracing. Different vector compression techniques are used to compress a given vector field while preserving its topology and main characteristics. This process produces a smaller data set, which is then used to calculate the trajectories of given particles. Lodah et al. [24], and Theisel et al. [25, 26] examined 2D spatial compression methods for reducing the data at each time step of the simulation. Tong et al. [27] explored temporal compression methods for reducing the number of time steps from N to M. Agranovsky et al. [28] explored determining which vector to retain based on priority, and then interpolating new vectors from the subset stored.

Vector field compression reduces the computational burden because it reduces the data size. However, it introduces errors. Vector field compression typically focuses on the vector field’s main features, omitting its minor characteristics. This omission may result in inaccurate interpolations when tracing particles’ trajectories.

2.2 Lagrangian-Based Particle Path Tracing.

Agranovsky et al. [2] explored a Lagrangian representation for flow study. Their approach had two phases: “in situ” and “post hoc.” Lagrangian basis flows were extracted from the simulation in the “in situ” stage, by seeding particles on a uniform grid at regular time intervals, and allowing them to exist in the simulation for a uniform constant time. At the end of a time interval, only the final position of each particle is stored to disk. The files generated contain Lagrangian basis flows instead of velocity vectors as in the traditional Eulerian method. These Lagrangian basis flows are then used to construct seeded particles’ trajectories.

In the “post hoc stage,” particles are arbitrarily seeded, and the uniform grid is then utilized to construct tetrahedrons around the particles. Nielson et al. [14] offer multiple tools for breaking cells into tetrahedrons. Tetrahedrons are chosen here because they can form the minimal possible convex envelope around a given point in a 3D space. Once the tetrahedrons are formed, the particles’ positions are expressed in terms of the tetrahedrons’ vertices. These vertices are used to determine the particles’ next positions. This process is repeated over a time interval in order to trace the journey of seeded particles over the given time interval.

Argranovsky et al. [2] found the Lagrangian approach to be more accurate, require less I/O, and be less computationally intensive compared to the traditional Eulerian representation. For a time that lies between two consecutive “time slices,” the traditional approach reads the vector field of the two “time slices” and interpolates between them to calculate the desired velocity. However, the Lagrangian approach will consider one file which covers a time interval of interest. This results in less files required in the Lagrangian method, which translates into less I/O. In the traditional approach, the interpolation between the two consecutive “time slices” introduces error because the time of interest is not saved to disk. The Lagrangian approach is more accurate because it considers “time intervals” as opposed to “time slices.” Additionally, it is less computationally intensive because it is not performing a numerical integration as in the traditional Eulerian approach.

Chandler et al. [3] use K-d Trees as the search structure for performing particles’ paths tracing from Lagrangian basis flows. In this approach, Lagrangian basis flows extracted from the simulation are defined as a mapping $P\colon \mathbb{R}\mathrm{x}\mathbb{N} \rightarrow\ \mathbb{R}^3$. A particle i at time t (i.e (i, t) in $\mathbb{R}\mathrm{x}\mathbb{N}$) is mapped to a 3D location (x_i, y_i, z_i). In the process of constructing the particles’ trajectories, the data from the time-step in consideration is loaded and used to build a K-d Tree. This K-d Tree is then used to search a given particle’s neighbors in order to perform the required interpolation in order determine the particle’s next position.

2.3 Introduction to Bounding Volume Hierarchy

BVHs are used in collision detection [17, 18] and ray tracing [16] as acceleration search structures. Similar to K-d Trees, BVHs are trees constructed from a data set. This type of tree is built such that every child is contained in its parent, and every node has a tight bounding box, which leads to BVHs having no empty nodes as opposed to K-d Trees [15]. One challenge in building BVHs, and even K-d Trees, comes from deciding where to split a given node in order to create its children. Different approaches, including Spatial Median [19, 20] and Surface Area Heuristic [21], have been used to effectively choose the split location. With the Spatial Median techniques, the split location is chosen to be the middle of the node’s bounding box, whereas the Surface Area Heuristic cost uses a formula to determine the optimal split with the minimum possible number of overlaps.

Stich et al. [4] compared multiple variants of BVHs and K-d Trees and determined that BVHs perform significantly better than K-d Trees for ray tracing. As opposed to K-d Trees, BVHs construct trees which often have fewer nodes, and no empty nodes. Because K-d Trees' construction does not yield nodes with tight bounding boxes, the resulting tree may end up with empty nodes. This difference allows BVHs to gain in performance and memory usage because they have less nodes to search from and to store in memory.

In our project, we consider the problem of searching for Lagrangian basis flows. We extend the work by Chandler in several key ways. First, we consider BVHs in addition to K-D Trees. Second, our study considers a spatiotemporal situation, which helps mitigate the problems related to temporal sparsity. Third, we consider a situation where the search structure in constructed once at the beginning, as opposed to periodic construction at each step.

3. STUDY

3.1 Configuration

We focused our work on how to find the neighbors of given particles using underlying Lagrangian basis flows extracted from a flow vector field. We developed a routine to generate basis flows, which is then used to build the search structures. We used the Arnold-Beltrami-Childress (ABC) [23] flow, a time dependent 3D vector field, to generate the data. We chose A = $\sqrt{3}$, B = $\sqrt{2}$, and C = 1.

$$\frac{dx}{dt} = A\sin(z) + C\sin(y) \tag{2}$$

$$\frac{dy}{dt} = B\sin(x) + A\sin(z) \tag{3}$$

$$\frac{dz}{dt} = C\sin(y) + B\sin(x) \tag{4}$$

Where the particle position is defined by P(x, y, z, t) and A, B, and C are constants. We generated multiple data sets from this vector field, varying the basis flows.

An important consideration for building a search structure is whether the input geometry is subdivided to optimize searching. We refer to this process as "primitives split." Incorporating this idea, we then built four spatiotemporal search structures to evaluate the different search methods for finding the neighbors of given particles. The four search structures are:

- K-d "with no primitive split"
- K-d "with primitive split"
- BVH "with no primitive split"
- BVH "with primitive split"

The construction of the different search structures requires the identification of a split location at each step. For each node, the split location is used to determine the children's bounding boxes. To compute the split location, we explore three different methods:

- Primitives Median Split
- Spatial Median Split
- Surface Area Heuristic cost / Volume Heuristic cost

In our work we'll refer to Surface Area Heuristic as Volume Heuristic instead because we are working with a 4D space as opposed to a 3D space.

3.2 Best Split Identification

With the Primitives Median [19, 20] approach, primitives are sorted along one of the axes. In our work, we chose the axis with the largest range, and the median primitive is picked as the split location. The Primitives Median often yields a balanced tree because the median primitive is always picked. However, it does not ensure fast search because this approach does not take into account the density of the Lagrangian basis flows. For very skewed data, this approach may result in poor performance. That said, the Primitive Median split yields an optimal split location when applied to a roughly uniform distribution of Lagrangian basis flows.

Sorting primitives is not necessary in the Spatial Median [19, 20] approach. For a given node, the different axes are compared and the middle of the axis with largest range is chosen to be the split location. The Spatial Median does not explicitly incorporate the density distribution in the decision of the split location. However, because space itself is the parameter considered in finding the split, it returns a better density distribution compared to the Primitives Median approach. This method does not yield a balanced search structure in the case of non-uniformly distributed Lagrangian basis flows. Additionally, finding the Spatial Median is performed in a constant time because sorting is not required. This results in a faster construction of the search structure. Similar to the Primtives Median techniques, the Spatial Median technique returns an optimal split location if the Lagrangian basis flows exhibit a roughly uniform distribution.

The Surface Area Heuristic [21] uses a probabilistic approach to identify the cost of performing a split at a specific location. In ray tracing, it determines the probability that a given ray will intersect with a bounding box [5]. This method is used to pick the split location which minimizes the number of overlaps with primitives. In our work, we are using a Volume Heuristic formula which is adapted from the Surface Area Heuristic formula, since we are dealing with 4D space instead of 3D.

$$Cost \;=\; \frac{V(L)\times N(L)+V(R)\times N(R)}{V} \tag{5}$$

Where V(L) represents the volume of the left bounding box, V(R) represents the volume of the right bounding box, N(L) represents the number of nodes that overlap with left bounding box, and N(R) represents the number of nodes that overlap with the right bounding box. V represents the node's volume heuristic.

In order to determine the best split, the different possible split locations are considered and their costs are calculated. This process is performed for all four axes. The split with the minimal cost is kept as the optimal split location. This method is expensive, but it yields a good split location. This technique, as opposed to the others, is built not only to minimize the number of overlaps with each split location, but also to include the density distribution in the split decision. It intrinsically uses the 4D space and the number of primitives to identify the optimal split location. Therefore, the Volume Heuristic approach exhibits characteristics that may contribute in building an optimal search structure.

3.3 K-d Trees Search Structures

As mentioned earlier, we built K-d Trees, which we compared with BVHs. We built two variants of K-d Trees. Both are constructed by first identifying the split location, which uses the three approaches described in section 3.2. The search structure is recursively built starting from the root and continuing until reaching the leaves. The termination criterion is determined by a threshold. Here the threshold is set to be a minimum number of primitives at the leaves.

3.3.1 K-d Tree "with no primitive split"

This K-d Tree does not incorporate primitive splitting. After the split location is found, iteratively, primitives' IDs of a given node are copied into their left child and/or right child depending on whether or not they overlap with the children's bounding boxes. This method generates several duplicate IDs in the resulting tree. The large number of duplicates not only increases memory footprint, but also incurs performance overhead. The duplicates need to be stored and taken into account when performing a search. Below is our implementation of a K-d tree "with no primitive split."

K-d tree "with no primitive split split"

```
load data from file
create root node

build(node)
   findBestSplit(node)
   performSplit(node)
        for each primitive
           if(overlaps with split location)
                Copy its ID in the left_child and right_child
           else
                Copy its ID in the left_child or right_child
        end for
   if(left_child is not a leaf)
        build(left_child)
   if(right_child is not a leaf)
        build(right_child)
```

3.3.2 K-d Tree “with primitive split”

Paragraph: how K-d tree with splits is implemented
The K-d tree “with primitive split” uses a very similar method to the K-d tree “with no primitive split”. For a given node, the split location is identified using the techniques described in section 3.2. Then, the node is divided at the split location creating its left and right child. Iteratively, the primitives’ ID’s are copied into the child they overlap with once the children are created. While iterating, the primitives which overlap with the split location are split into two new primitives. The data is updated with the new primitives and their IDs are copied into their matching children. This process is done recursively starting with the root node. The K-d Tree “with primitive split” generates less ID duplicates compared to the K-d Tree “with no primitive split.” Below is our implementation of a K-d Tree “with primitive split.”

K-d tree “with primitive split split”

```
load data from file
create root node

build(node)
   findBestSplit(node)
   performSplit(node)
          for each primitive:
             if(overlaps with split location)
                    Split primitives into 2 primitives
```

```
9                 Update data with new primitives
10                Copy new primitive 1 ID in left_child
11                Copy new primitive 2 ID in righ_child
12            else
13                Copy its ID in the left_child or right_child
14        end for
15    if(left_child is not a leaf)
16        build(left_child)
17    if(right_child is not a leaf)
18        build(right_child)
```

3.4 BVHs Search Structures

Both versions of K-d Trees described above do not take into account the notion of tight bounding boxes, which is the primary difference between BVHs and K-d Trees. Similar to the K-d Trees, we implemented two versions of BVHs: Spatial Split BVH [4] which we refer to as a BVH “with primitive split,” and regular BVH, which we refer to as BVH “with no primitive split.” Again they both utilize the techniques in section 3.2 to choose the split location, and they are built recursively.

3.4.1 BVH “with no primitive split”

BVH [21] “with no primitive split” has traditionally been used in ray tracing as an optimization over K-d Trees. For a given node, building a BVH requires the identification of the best split location. The split location is then used to construct the node’s children. During the process of splitting the node, for the primitives which overlap with the split location, their IDs are copied into the children which contain their center. After each split, the children bounding boxes are adjusted in order to keep them tight with respect to the primitives they contain. The recursive construction of this search structure is initiated from the root node, which contains all the IDs. This approach yields tightly bounded nodes and no duplicates between siblings. Because no splitting is performed, the size of the data remains constant during the process. However, the bounding boxes of siblings may overlap. Below is the description of our implementation of BVH "with no primitive split".

BVH "with no primitive split split"

```
load data from file
create root node

build(node)
  findBestSplit(node)
  performSplit(node)
      for each primitive
       if (center <= split location)
             Copy its ID in the left_child
             Update left_child bounding box
        else
             Copy its ID in the right_child
             Update right_child bounding box

      end for
  if(left_child is not a leaf)
      build(left_child)
  if(right_child is not a leaf)
      build(right_child)
```

3.4.2 BVH “with primitive split”

Spatial Split BVH [4] is similar to regular BVH, except that primitives which overlap with the split location are split in two. As in all the previous structures, the split location for a given node is identified using the methods described in section 3.2. The node split location is then used to construct its children. In the process of splitting the node, the primitives which overlap with the split location are also split. The data is updated with the new primitives generated from the splitting, and the IDs of the primitives are copied into the children they overlap with. The Spatial Spilt BVH returns a tree with nodes which are tightly bounded, along with no overlaps among siblings.

BVH “with primitive split split”

```
load data from file
create root node

build(node)
  findBestSplit(node)
  performSplit(node)
       for each primitive:
        if(overlaps with the split location)
              Split primitives into 2 primitives
              Update data with new primitives
              Copy new primitive 1 ID in left_child
              Update left_child bounding box
```

```
12                  Copy new primitive 2 ID in righ_child
13                  Update right_child bounding box
14              else
15                  Copy its ID in the left_child or right_child
16                  Update left_child or right_child bounding box
17          end for
18      if(left_child is not a leaf)
19          build(left_child)
20      if(right_child is not a leaf)
21          build(right_child)
---------------------------------------------------------------------
```

3.5 Search

After we built the search structures, we constructed a search routine to find the neighbors of given particles. The routine has two main steps. In the first step, we recursively searched our structure, K-d Tree or BVH, for all the smallest possible nodes which overlaps with our search criteria. The search parameters consisted of a point P(x, y, z, t) and a radius r. In the second step, we looped through the resulting nodes obtained from the first step and extract all the primitives' IDs which met the constraints imposed by the search parameters. This search routine is used for all our different search structures. Below is a brief algorithm which describes the search steps.

```
---------------------------------------------------------------------
                              Search
---------------------------------------------------------------------

1       findOverlappedNode (node, P, r)
```

```
2        if(P is in left_child)
3            findOverlappedNode(left_child, P, r)
4        else
5            save node in result
6        if(P is in right_child)
7            findOverlappedNode(right_child, P, r)
8        else
9            save node in result

10   extractPrimitives(result)
11       for each node in result:
12           if(getRadius(P) < r)
13               save primitive in final_result
------------------------------------------------------------------------
```

4 EXPERIMENTAL OVERVIEW

4.1 Experiment Factors

Our experiment consisted of comparing the different search structures, in order to inform the optimal spatiotemporal search structure for particle path tracing from Lagrangian basis flows. Our experiments had three key varying parameters: the method used to determine the split location, the type of search structure, and the number of Lagrangian basis flows. For determining the split location, we utilized the Spatial Median, Primitives Median, and Volume Heuristic approaches which are described in section 3.2. For the search structures, we focused on the two variants of K-d Trees and two variants of BVHs. For the number of Lagrangian basis flows we used three different sets from ABC data. The cross product of these different parameters gives us

- 3 x 4 x 3 = 48 experiments.

i.e., 3 split identification methods, 4 search structures, and 3 sets of ABC Lagrangian basis flows.

4.2 Evaluation Criteria

In this work, we investigate different approaches for finding neighboring Lagrangian basis flows when given a spatiotemporal coordinate. More specifically it examines the different spatiotemporal search structures with the following three factors:

- Storage Overhead: The overhead to represent the search data structure is very important because it relates to the memory usage. Significant overhead leads to higher memory accesses and increased memory to store the Lagrangian basis flows and the search structures.

- Build time: The time it takes to build the search structures is a one-time cost amortized by the many searches. That said, high build times can be problematic.

- Search time: The primary focus of our work is to evaluate the time it takes to search the neighbors of a given particle. This is important because in the process of tracing the trajectory of a particle, the search for neighbors is required at every step.

4.3 Measurements

Our methodology focuses on making various measurements of the different key aspects of the search structures. These key aspects play a vital role in the performance of the search structures. The key characteristics measures are:

- Number of primitives and number of nodes
- Size of the search structures and Lagrangian basis flows
- Build time of the search structures
- Search time for finding neighbors

4.3.1 Number of primitives and number of nodes

The increase in number of primitives and/or nodes is directly related to the increase in data size for the Lagrangian basis flows and the search structures. We compared the number of primitives before and after the building of the search structures. We also compared the number of primitives and nodes among the various search structures.

4.3.2 Size of search structures and data

Often, large memory is required to store the Lagrangian basis flows and the search structures. We analyzed the memory usage by comparing the data size before and after the building of the search structures. In addition, we compared the data size among the different search structures.

4.3.3 Build time

Build time is the time, in seconds, that it takes to build a search structure given a set of Lagrangian basis flows. There are two main factors that dictate the build time: the time it takes to find the split location and the time it takes to perform the actual split on the node. We measured the time it takes to build the different search structures and compared them to each other.

4.4.4 Search Time

Search time is the time, in seconds, that it takes to find the neighbors of a given particle using the underlying Lagrangian basis flows. The search time is affected by the number of nodes visited when performing a search, and the time it takes to obtain the matching primitives from the nodes found during the traversal. We measured the average number of visits and the search time of the various search structures, and then compared them to each other.

4.4 Machine

Our experiments were conducted on a machine using an Intel(R) Core(TM) i-7 with a frequency of 3.6GHz. The memory size is 16 GB and it runs at frequency of 1600 MHz.

5. RESULTS

This section discusses the results obtained from the different experiments.

5.1 Number of Primitives and Nodes

Tables 1, 2, 3, and 4 show the show the sizes, in terms of the number primitives and nodes, for K-d Tree and BVH. These sizes are directly related to the amount of storage required. For small numbers of Lagrangian flows, the K-d Tree “with primitive no split” coupled with Primitive Median split yields the smallest number of primitives in its output. As the number of primitives increases, the BVH “with primitive split” coupled with the Volume Heuristic yields the best result. With 10,000 Lagragian basis flows, the number of primitives and nodes for the BVH “with primitive split” are half the the number of primitives and node of the K-d Tree “with primitive split.”

K-d Tree “with no primitive split”			
Split type	# Lagrangian basis flows	Final number of primitives	Number of nodes
Spatial Median	100 1000 10000	4777 49100 0.51 10^6	375 22117 10^6
Primitive Median	100 1000 10000	4777 49100 0.51 10^6	59 711 16765
Volume Heuristic	100 1000 10000	4777 49100 0.51 10^6	113 971 10^6

Table 1: Results related to K-d Tree “with primitive no split.” This table shows the number of primitives and the number of nodes given different Split methods and number of Lagrangian basis flows.

K-d Tree “with primitive split”				
Split type	# Lagrangian basis flows	Number of input primitives	Final number of primitives	Number of nodes
Spatial Median	100 1000 10000	4777 49100 $0.51\ 10^6$	8444 $0.90\ 10^6$ $14.98\ 10^6$	375 20263 112563
Primitive Median	100 1000 10000	4777 49100 $0.51\ 10^6$	5671 90103 1.55 M	65 1035 9559
Volume Heuristic	100 1000 10000	4777 49100 $0.51\ 10^6$	5544 $0.11\ 10^6$ $1.94\ 10^6$	21 635 12621

Table 2: Results related to K-d Tree “with primitive split.” This table shows the number of primitives and the number of nodes given different Split methods and number of Lagrangian basis flows.

BVH “with no primitive split”			
Split type	# Lagrangian basis flows	Final number of primitives	Number of nodes
Spatial Median	100 1000 10000	4777 49100 $0.51\ 10^6$	219 1999 12227

Primitive Median	100	4777	63
	1000	49100	481
	10000	0.51 10^6	7761
Volume Heuristic	100	4777	73
	1000	49100	537
	10000	0.51 10^6	16159

Table 3: Results related to BVH “with primitive no split.” This table shows the number of primitives and the number of nodes given different Split methods and number of Lagrangian basis flows.

BVH “with primitive split”				
Split type	# Lagrangian basis flows	Number of input primitives	Final number of primitives	Number of nodes
Spatial Median	100	4777	8456	161
	1000	49100	1.30 M	19097
	10000	0.51 10^6	19.43 10^6	90515
Primitive Median	100	4777	7888	103
	1000	49100	1.09 10^6	15117
	10000	0.51 10^6	1.63 10^6	80453
Volume Heuristic	100	4777	5867	95
	1000	49100	70968	539
	10000	0.51 10^6	880979	3771

Table 4: Results related to BVH "with primitive split." This table shows the number of primitives and the number of nodes given different Split methods and number of Lagrangian basis flows.

5.2 Storage Overhead

Tables 5, 6, 7, and 8 show the size of memory, in bytes, required to store the Lagrangian basis flows and the search structures. For a small amount of Lagrangian basis flows, the K-d Tree "with no primitive split" coupled with Volume Heuristic, requires the smallest amount of memory. However, as the number of Lagrangian basis flows becomes larger the BVH "with no primitive split" coupled with the Volume Heuristic split method requires the least amount of storage for the Lagrangian basis flows and the search structures. With 10,000 Lagrangian basis flows, the of storage required for BVH "with no primitive split" coupled with the Volume Heuristic is almost 2X the storage required for BVH "with primitive split," 4X the storage required for K-d Tree "with primitive split," and 9X the storage required for K-d Tree "with no primitive split,"

K-d Tree "with no primitive split"				
Split type	# Lagrangian basis flows	Storage of Lagrangian flows in bytes	Structure size in bytes	Total data size in bytes
Spatial Median	100 1000 10000	0.65 M 6.68 M 69.04 M	0.42 M 33.41 M 952.41 M	1.07 M 40.09 M 10^3 M

Primitive Median	100 1000 10000	0.65 M 6.68 M 69.04 M	0.13 M 2.23 M 50.31 M	0.78 M 8.91 M 119.35
Volume Heuristic	100 1000 10000	0.65 M 6.68 M 69.04 M	0.25 M 3.80 M 843.48 M	0.90 M 10.35 M 912.52 M

Table 5: Results related to K-d Tree “with primitive no split.” This shows the storage size required for Lagrangian basis flows and the search structures.

K-d Tree “with primitive split”				
Split type	# Lagrangian basis flows	Storage of Lagrangian flows in bytes	Structure size in bytes	Total data size in bytes
Spatial Median	100 1000 10000	1.15 M 121.97 M 69.04 M	0.61 M 140.85 M 345.00 M	1.76 M 262.82 M 414.04 M
Primitive Median	100 1000 10000	0.77 M 12.25 M 205.85 M	0.15 M 3.82 M 83.13 M	0.92 M 16.09 M 288.98 M
Volume Heuristic	100 1000 10000	0.75 M 14.58 M 256.85 M	0.09 M 5.17 M 149.54 M	0.84 M 19.75 M 406.39

Table 6: Results related to K-d Tree "with primitive split." This shows the storage size required for Lagrangian basis flows and the search structures.

BVH "with no primitive split"				
Split type	# Lagrangian basis flows	Storage of Lagrangian flows in bytes	Structure size in bytes	Total data size in bytes
Spatial Median	100 1000 10000	0.65 M 6.68 M 69.04 M	0.32 M 14.16 M 344.69 M	0.97 M 21.02 M 413.73 M
Primitive Median	100 1000 10000	0.65 M 6.68 M 69.04 M	0.12 M 1.80 M 27.02 M	0.77 M 8.48 M 96.06 M
Volume Heuristic	100 1000 10000	0.65 M 6.68 M 69.04 M	0.17 M 2.36 M 59.89 M	0.82 M 9.04 M 102.93 M

Table 7: Results related to BVH "with primitive split." This shows the size of storage required for Lagrangian basis flows and the search structure.

BVH "with primitive split"				
Split type	# Lagrangian basis flows	Storage of Lagrangian flows in bytes	Structure size in bytes	Total data size in bytes

Spatial Median	100 1000 10000	1.15 M 177.32 M 2.15 10^3 M	0.34 M 120.29 M 1.78 10^3 M	1.49 M 297.61 M 3.93 10^3 M
Primitive Median	100 1000 10000	1.07 M 147.35 M 2.15 10^3 M	0.23 M 65.80 M 1.08 10^3 M	1.30 M 213.15 3.23 10^3 M
Volume Heuristic	100 1000 10000	0.80 M 9.65 M 119.81 M	0.24M 3.58 M 54.19 M	1.04 M 13.23 M 174.00 M

Table 8: Results related to BVH "with primitive split." This shows the size of storage required for Lagrangian basis flows and the search structure.

5.3 Build Time

As mentioned previously, the search structure build time is dependent on the time it takes to find the split location and the time it takes to perform a split on a given node. The Spatial Median approach for finding a split location does not require any sorting as opposed to the other approaches. Tables 9, 10, 11, and 12 show that the Primitive Median approach yields a shorter build time compared to the Volume Heuristic approach. The Volume Heuristic approach considers multiple locations when calculating the split location as opposed to the other two. These results match our expectations.

K-d Tree "with no primitive split"		
Split type	# Lagrangian basis flows	Build time in sec

Spatial Median	100	0.009
	1000	0.445
	10000	12.932
Primitive Median	100	0.007
	1000	0.120
	10000	1.829
Volume Heuristic	100	0.081
	1000	1.19
	10000	138.55

Table 9: Results related to K-d Tree “with primitive no split.” This table shows the time it takes to build the different search structures using the various split location methods.

K-d Tree “with primitive split”		
Split type	# Lagrangian basis flows	Build time in sec
Spatial Median	100	0.010
	1000	0.713
	10000	11.900
Primitive Median	100	0.008
	1000	0.142
	10000	1.940
Volume Heuristic	100	0.098
	1000	0.899

	10000	20.54

Table 10: Results related to K-d Tree "with primitive split." This table shows the time it takes to build the different search structures using the various split location methods.

BVH "with no primitive split"		
Split type	# Lagrangian basis flows	Build time in sec
Spatial Median	100 1000 10000	0.015 4.48 7.190
Primitive Median	100 1000 10000	0.017 0.101 1.171
Volume Heuristic	100 1000 10000	0.382 4.476 126.045

Table 11: Results related to BVH "with primitive split." This table shows the time it takes to build the different search structures using the various split location methods.

BVH "with primitive split"		
Split type	# Lagrangian basis flows	Build time in sec

Spatial Median	100	0.013
	1000	0.896
	10000	11.57
Primitive Median	100	0.021
	1000	0.760
	10000	11.707
Volume Heuristic	100	0.080
	1000	0.922
	10000	9.758

Table 12: Results related to BVH “with primitive split.” This table shows the time it takes to build the different search structures using the various split location methods.

5.4 Search Time

Tables 13, 14, 15, and 16 show the average number of visits and the average search time. The smallest possible search time is desirable because the search is performed at every step when tracing a particle trajectory. The BVH “with primitive split” using the Volume Heuristic approach for determining the split location yields a shorter search time and a smaller average number of traversals compared to the others.

K-d Tree “with no primitive split”			
Split type	# Lagrangian basis flows	Av number of node visits	Av search time in sec

Spatial Median	100	4.91	0.020
	1000	5.36	0.030
	10000	4.20	8.383
Primitive Median	100	5.00	0.028
	1000	10.13	0.032
	10000	13.51	0.049
Volume Heuristic	100	14.84	0.023
	1000	3.22	0.023
	10000	2.28	0.607

Table 13: Results related to K-d Tree "with primitive no split." This table shows the average number of traversals and the average build time for the different search structures.

K-d Tree "with primitive split"			
Split type	# Lagrangian basis flows	Av number of node visits	Av search time in sec
Spatial Median	100	4.92	0.028
	1000	5.29	0.196
	10000	6.46	3.00
Primitive Median	100	4.20	0.050
	1000	9.76	0.048
	10000	16.15	0.187

Volume Heuristic	100 1000 10000	5.06 21.78 141.73	0.035 0.167 3.626

Table 15: Results related to K-d Tree “with primitive split.” This table shows the average number of traversals and the average build time for the different search structures.

BVH “no with primitive split”			
Split type	# Lagrangian basis flows	Av number of node visits	Av search time in sec
Spatial Median	100 1000 10000	11.23 5.23 66.33	0.019 0.023 0.050
Primitive Median	100 1000 10000	4.23 5.13 12.14	0.020 0.022 0.040
Volume Heuristic	100 1000 10000	12.04 5.23 11.59	0.023 0.023 0.067

Table 15: Results related to BVH “with primitive no split.” This table shows the average number of traversals and the average build time for the different search structures.

BVH "with primitive split"			
Split type	# Lagrangian basis flows	Av number of node visits	Av search time in sec
Spatial Median	100 1000 10000	3.79 2.73 66.34	0.018 0.019 0.051
Primitive Median	100 1000 10000	4.25 6.75 10.96	0.020 0.020 0.038
Volume Heuristic	100 1000 10000	16.19 2.99 4.08	0.023 0.021 0.033

Table 16: Results related to BVH "with primitive split." This table shows the average number of traversals and the average build time for the different search structures.

5.5 Result Summary

In the previous sections, we made isolated observations about the different characteristics measured. In this section, we discuss the different connections that exist among the different features utilized to build the search structures. Table 9 shows the summary of the measurements made for 10,000 Lagrangian basis flows. These results indicate that a search structures built with Volume Heuristic + BVH has a smaller storage overhead, shorter search time, and longer build time compared to any of the other search structures. These results show that the time invested in building the search structures can be amortized with multiple searches when using the Volume Heuristic approach. This informs that BVH aids to reduce storage overhead and improve search time.

Search Structure	Data size in bytes	Build Time in sec	Search Time in sec
Spatial Median + K-d Tree “with primitive split”	414.04 M	11.900	3.007
Spatial Median + BVH “with primitive split”	3.93 10^3 M	11.570	0.051
Primitive Median + K-d Tree “with primitive split”	288.98 M	1.940	0.187
Primitive Median + BVH “with primitive split”	3.23 10^3 M	11.707	0.038
Volume Heuristic + K-d Tree “with primitive split”	406.39 M	20.540	3.626
Volume Heuristic + BVH “with primitive split”	174.00 M	9.758	0.033

Table 9: Summary of different search structures’ results. This table shows the total data size in bytes to store search structures and 1,000 Lagrangian basis flows, the time it takes to build the search structures, and the time it takes to search the neighbors of a set of points given a radius.

6 FUTURE WORK AND CONCLUSION

This study indicates that the search structures using the Volume Heuristic approach coupled with the BVH “with primitive split” yields better performance than the other search structures explored. Various features contributed to the better performance observed with this search structure. We evaluated the search structures by considering three types of split methods: the Spatial Median, the Primitive Median, and the Volume Heuristic. These three split methods were coupled with four types of search structures: BVH “with primitive split”, BVH “with no primitive split,” K-d Tree “with primitive split,” and K-d Tree “with no primitive split.” For a given node, the Volume Heuristic approach chooses a split location which takes into account the number of primitives in its children and the density distribution of the primitives in 4D space. The BVH “with primitive split” yields a search structures with nodes that are tightly bounded, and no ID duplicates among sibling nodes.

In terms of future work, we should investigate hybrid versions of the search structures to further inform the choice of search structure for finding neighbors from Lagrangian flow basis. For instance, we could construct a search structure that uses both BVH “with no primitives” and BVH “with primitive split.” In addition, this work analyzed the serial versions of the different search structures. Thus, a similar analysis must be performed for parallelized versions of the search structures. This may lead to great performance gains in the build time and search time.

This project is part of a larger scope study, which encompasses the extraction of Lagrangian basis flows, the search for neighboring Lagrangian basis flows, and the interpolation to compute the particles’ trajectories. This study focused on finding the neighboring Lagrangian basis flows when given a spatiotemporal coordinate. Therefore, more investigations on how to extract the Lagrangian basis flows, and how to perform the interpolation are required to construct a full story around particle trajectory tracing from Lagrangian basis flows.

7 REFERENCES

[1] T. McLoughlin, R. S. Laramee, R. Peikert, F. H. Post, and M. Chen. Over Two Decades of Integration-Based, Geometric Flow Visualization. In EuroGraphics 2009 - State of the Art Reports, pages 73–92, April 2009.

[2] A. Agranovsky, D. Camp, C. Garth, E. W. Bethel, K. I. Joy and H. Childs, "Improved post hoc flow analysis via Lagrangian representations," *Large Data Analysis and Visualization (LDAV), 2014 IEEE 4th Symposium on*, Paris, 2014, pp. 67-75.

[3] J. Chandler, H. Obermaier and K. I. Joy, "Interpolation-Based Pathline Tracing in Particle-Based Flow Visualization," in *IEEE Transactions on Visualization and Computer Graphics*, vol. 21, no. 1, pp. 68-80, Jan. 1 2015.

[4] Martin Stich, Heiko Friedrich, and Andreas Dietrich, "Spatial splits in bounding volume hierarchies," in *Proceedings of the Conference on High Performance Graphics 2009*. ACM, 2009, pp. 7–13.

[5] David J. MacDonald and Kellogg S. Booth. 1990. Heuristics for ray tracing using space subdivision.*Vis. Comput.* 6, 3 (May 1990), 153-166

[6] B.Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pages 263–270, New York, NY, USA, 1993. ACM.

[7] G. Haller. Distinguished material surfaces and coherent structures in three-dimensional fluid flows. *Physica D: Nonlinear Phenomena*, 149(4):248 – 277, 2001.

[8] J. P. M. Hultquist. Constructing stream surfaces in steady 3d vector fields. In *Visualization, 1992. Visualization '92, Proceedings., IEEE Conference on*, pages 171–178, Oct 1992.

[9] J. R. Cash and A. H. Karp. A variable order runge-kutta method for initial value problems with rapidly varying right-hand sides. *ACM Trans. Math. Softw.*, 16(3):201–222, Sept. 1990.

[10] F.Sadlo ,A. Rigazzi, and R. Peikert. Time-dependent visualization of lagrangian coherent structures by grid advection. *Topological Methods in Data Analysis and Visualization*, pages 151–165, 2011.

[11] T. Salzbrunn, C. Garth, G. Scheuermann, and J. Meyer. Pathline predicates and unsteady flow structures. *The Visual Computer*, 24(12):1039–1051, 2008.

[12] G. Haller and G. Yuan. Lagrangian coherent structures and mixing in two-dimensional turbulence. *Physica D: Nonlinear Phenomena*, 147(3-4):352 – 370, 2000.

[13] G. Haller. Finding finite-time invariant manifolds in two-dimensional velocity fields. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 10(1):99–108, 2000.

[14] G. M. Nielson. Tools for triangulations and tetrahedrizations. In *Scientific Visualization, Overviews, Methodologies, and Techniques*, pages 429–525, Washington, DC, USA, 1997. IEEE Computer
Society.

[15] Michal Hapala and Vlastimil Havran, "Review: Kd-tree traversal algorithms for ray tracing," in *Computer Graphics Forum*. Wiley Online Library, 2011, vol. 30, pp. 199–213.

[16] Steven G Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, et al., "Op- tix: a general purpose ray tracing engine," *ACM Transactions on Graphics (TOG)*, vol. 29, no. 4, pp. 66, 2010.

[17] Thomas Larsson and Tomas Akenine-Möller, "Efficient colli- sion detection for models deformed by morphing," *The Visual Computer*, vol. 19, no. 2, pp. 164–174, 2003.

[18] Matthias Teschner, Stefan Kimmerle, Bruno Heidelberger,
Gabriel Zachmann, Laks Raghupathi, Arnulph Fuhrmann, M-P Cani, François Faure, Nadia Magnenat-Thalmann, Wolfgang Strasser, et al., "Collision detection for deformable ob- jects," in *Computer graphics forum*. Wiley Online Library, 2005, vol. 24, pp. 61–81.

[19] Brian Smits*, "Efficiency issues for ray tracing,"* Journal of Graphics Tools, 1998.

[20] Timothy L Kay and James T Kajiya, "Ray tracing complex scenes," in *ACM SIGGRAPH computer graphics*. ACM, 1986, vol. 20, pp. 269–278.

[21] Manfred Ernst and Günther Greiner, "Early split clipping for bounding volume hierarchies," in *Interactive Ray Tracing, 2007. RT'07. IEEE Symposium on*. IEEE, 2007, pp. 73–78.

[22] Christopher Johnson, Steven G Parker, Charles Hansen, Gordon L Kindlmann, and Yarden Livnat, "Interactive simulation and visualization," *Computer*, vol. 32, no. 12, pp. 59–65, 1999.

[23] T. Dombre, U. Frisch, J. M. Greene, M. Hénon, A. Mehr, and A. M. Soward (1986). "Chaotic streamlines in the ABC flows". *Journal of Fluid Mechanics*, 167, pp. 353–391 doi:10.1017/S0022112086002859

[24] S. Lodha, J. Renteria, and K. Roskin. Topology preserving compression of 2d vector fields. In *Visualization 2000. Proceedings*, pages 343–350, 2000.

[25] H. Theisel, C. Rossl, and H.P.Seidel.Combining topological simplification and topology preserving compression for 2d vector fields. In *Computer Graphics and Applications, 2003. Proceedings. 11th Pacific Conference on*, pages 419–423, 2003.

[26] H.Theisel, C.Ro¨ssl, and H.-P. Seidel. Compression of 2d vector fields under guaranteed topology preservation. *Computer Graphics Forum*, 22(3):333–342, 2003.

[27] X. Tong, T.-Y. Lee, and H.-W. Shen. Salient time steps selection from large scale time-varying data sets with dynamic time warping. In *Large Data Analysis and Visualization (LDAV), 2012 IEEE Symposium on*, pages 49–56, Oct 2012.

[28] A. Agranovsky, D. Camp, K. I. Joy, and H. Childs. Subsampling-Based Compression and Flow Visualization. In SPIE Conference on Visualization and Data Analysis (VDA), volume 9397, pages 93970J–01–93970J–14, San Francisco, CA, Feb. 2015.

[29] H. Childs. Data Exploration at the Exascale. Supercomputing Frontiers and Innovations, 2(3):5–13, Dec. 2015.