Optimisation of parallel KD-trees using heuristics for neuron touch detection task

Daniel Benedí García

Optimisation of parallel k-d trees using heuristics for neuron touch detection task

Daniel Benedí García

Degree Project in Computer Science

Date: 18th of May of 2022

Supervisor: Alexander Kozlov

Examiner: Pawel Herman

School of Electrical Engineering and Computer Science

Swedish title: Optimering av parallell k-d-träd med hjälp av heuristik för neuronberöring upptäcktsuppgift

Abstract

Neuroscience has benefited from neuronal network simulation and an important task in the simulation is finding points in space where two neurites approach each other so a synapse could be formed. The task of finding the touching points could be seen as similar to the ray collision in ray tracing in computer graphics. This thesis aimed to investigate if the heuristics used in computer graphics and self-defined to speed up ray tracing can be used in the neurite touchpoint task. For analysis, we measured the time used for building the k-d trees (one per neuron), the time for querying and the memory usage. The tests were made using one specific neuron type called interneuron and realistic densities. This was made for simplicity, but the only difference with other types of neurons is the conditions for generating a touching point. It was found that due to their density, the interneurons do not benefit from these heuristics.

Sammanfattning

Neurovetenskap har tagit nytta av neurala nätverkssimulering och en viktig uppgift i simuleringar är att hitta punkter i rymden där två neuriter närmar sig varandra så att en synaps kan bildas. Uppgiften att hitta beröringspunkter påminner om strålkollision vid strålspårning i datorgrafik. Denna studie syftar till att undersöka huruvida heuristiken som används i datorgrafik för att påskynda strålföljning kan appliceras i neuron beröringspunktsproblem. För analys mätte vi tiden som tar för att bygga en algoritm med namnet kdtrees (en per neuron), tid för beräkningar, samt minnesanvändning. Testerna gjordes genom att använda en specifik neurontyp som kallas för interneuron och realistiska tätheter. Interneuroner är valda för att förenkla metodiken i studien, men den enda skillnaden mellan interneuron och andra typer av neuroner är tillståndet att generera en beröringspunkt. Resultaten visar att på grund av neurondensitet, en interneuron får ingen nytta av heuristiken i datorgrafik.

Contents

Chapter 1

Introduction

In the last decade, neuroscience has benefited from simulating nerve cell behaviour or neural networks. The simulation made use of large-scale and high-precision experimental methods of data acquisition to develop fact-based tools that helped understand brain functions and diseases [25]. Due to the amount of data collected, the simulations are done with supercomputers integrating different levels of simulation.

A human brain consists of 100 billion neurons [14] of different types depending on their function, shape and other factors. In physiology, a neuron consists of several parts and from a morphological point of view, these parts, also called compartments, form a tree. Therefore, every compartment has only one parent and can have several children, typically no more than 2. When two cells are close enough, also called they have a touchpoint, they can develop a synapse. A synapse is a structure that permits a neuron to pass a signal to another neuron.

One usual workflow consists of first doing a single-cell morphology reconstruction, then building a local network finding touchpoints and selecting possible synapses [30] and finally simulating the signals exchange. Usually, a neuron has 1000 inputs that fire at an average rate of 10 Hz and each connection requires around ten instructions. These make the simulation in real-time of a human brain infeasible even with terascale computers.

When building a local network, space-partitioning structures are used for faster searches of touchpoints. This thesis will consider k-d trees as the space-partitioning structure as previous research showed it performs better for this task[1]. A Kd-tree is a space-partitioning data structure for organizing points in a k-dimensional space by dividing the

Euclidian space into two convex sets using hyperplanes. The method that divides the space into two subsets is called binary space partitioning (BSP). Other examples of BSP are Octrees and Quadtrees. Some applications of BSP are robotics, computer graphics, and computer-aided design.

The field of computer graphics uses a technique for rendering a scene called ray-tracing. This technique models light transport along the stage to get the image. Although ray-tracing generates realistic images, it has a high computational cost. Computer scientists developed several techniques to reduce the high computational cost, for instance, BSP [12]. Also, they introduced the usage of kd-trees and heuristics to improve the performance [5, 16, 17, 23, 37].

1.1 Problem statement

This project aims to compare different heuristics used in computer graphics concerning performance. We will measure the performance of the heuristics in terms of execution time for building the kd-trees and for querying and random access memory (RAM) usage. These measurements will occur during typical scenarios of data-driven reconstruction of a neural network. Therefore, this thesis aims to investigate the following:

How does the usage of heuristics in Kd-trees affect the performance of touch dectetion in neuronal morphometrics?

1.2 Scope and delimitations

This thesis will focus on analyzing the performance of four heuristics in Kd-trees: Surface Area Heuristic (SAH) and Curve Complexity Heuristic (CCH), these two are common in computer graphics, and the other two are our own proposal, the median of the hyperplane with maximum variance split and minimum variance union split. We test the Kd-trees data structure in serial and parallel execution.

We use a specific type of neuron called a fast-spiking interneuron to evaluate the performance. Fast-spiking interneuron allows every kind of synapse, not only axodendritic synapsis. We evaluate two different cases:

- Given a realistic density, try a different number of neurons.
- Given a certain number of neurons, try different densities.

Furthermore, we will try to extrapolate with densities larger than the commons for interneurons to test how these heuristics will affect the performance of different neurons.

However, this thesis will perform all measurements on a personal computer. Due to the project's time frame, all the desired measurement will not be possible.

1.3 Thesis outline

The background (Chapter 2) presents the used data structure and relevant algorithms as well as the libraries used in this thesis. The methods (Chapter 3) explain in detail how measurements were made and the motivations for choices made. The measurements are then presented in the result chapter (Chapter 4) with the corresponding analysis. Subsequently, the results are discussed in the discussion chapter (Chapter 5) and a conclusion is presented (Chapter 6).

Chapter 2

Background

2.1 Neuronal morphology

Neurons are the fundamental units of the nervous system. There are different types of neurons depending on their function, shape and other factors. If we classify by function, there are three types: sensory neurons, motor neurons and interneurons. The sensory neurons respond to stimuli and send signals to the spinal cord or brain. The motor neurons receive information from the brain and spinal cord to control everything, like muscles or glands. Interneurons connect other neurons within the same region. A neural circuit or neuronal network is multiple neurons connected.

In physiology, a neuron consists of a cell body or soma with the nucleus, dendrites and axon, as seen in the figure 2.1.1. The soma is a compact structure, whereas the axon and dendrites are filaments from the soma. Typically a dendrite receives signals from other neurons and the axon transmits information to other neurons. The cell body, soma, is where the nucleus lies and where proteins are made to be transported throughout the axon and dendrites.

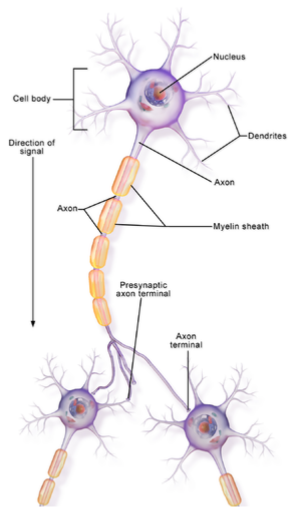

Figure 2.1.1: A sketch of how a neuron may look like [29]

A point where the distance between two neurons is lower than a certain threshold defines a touchpoint. If there is a touchpoint, then a synapse can emerge. A synapse is a structure that permits a neuron to pass an electrical or chemical signal to another

neuron. There are different types of synapses. The most common
is axodendritic. But there are also other types, depending on which parts of the neurons are in touch, such as axo-axonic (axon with axon), dendro-dendritic (dendrite with dendrite), axo-secretory (axon to bloodstream), somatodendritic (soma to dendrite), dendro-somatic (dendrite with soma), and somato-somatic (soma with soma) synapses. A neuron can have up to 6000 synapses with neighbouring neurons[33].

2.2 Simulation of local neuronal network

The usual workflow in research consists of three main steps: single-cell morphology reconstruction, building a local network and simulating the signals exchange.

The single-cell morphology reconstruction consists of retrieving the neurons' morphology to simulate. To do so, first, the bunch of neurons need to be separated to be able to do a reconstruction of the morphology of the neurons and a classification. Also, a repairing phase is necessary because there could have been errors in previous steps. This process ends up with isolated single-cell models which basically is a file with 3D points in space with their parent, extra data and some classification like dendrite or axon. There are different databases with reconstructions of neuron morphologies, both private and public access, but the most common and huge is NeuroMorpho.Org [3, 4].

The network building step is computationally more intensive because there is a large number of points in the space where a lot of spatial calculations are involved. First, the single-cell models are cloned, transformed and translated to their final position. Then, all the touching points between the neurons need to be found. This task has a higher computational cost because it implies checking almost all the points of the neurons against all the compartments of the other neurons. Finally, the synapses are established according to geometric and biological restrictions.

Finally, simulating the signal exchange of the neurons relies on different software tools such as NEURON [8], Arbor [2], Brian [34] and IBM Neural Tissue Simulator [22]. These programs allow the simulation of detailed biophysical models from single-cell models to large-scale networks.

2.3 Space-partition data structures

Space partitioning is the process of dividing a space into disjoint subsets, non-overlapping. The division can be done in two or more regions, and any point can then lie in only one subregion. Space partitioning is often hierarchical, which means the subsets are also divided recursively into other areas. The recursive division allows the organization of the subsets into a tree called a space-partitioning tree.

The space-partitioning trees can divide the subsequent space into several regions each time or just two each iteration. When splitting the area into only two subregions, it is called binary space partitioning (BSP) tree. BSP trees usually use a hyperplane, a generalization of the plane for higher dimensions, to divide space, so the points on each side of the hyperplane form each subregion.

There is also another kind of tree that not only does it divides the space but also the time. For example, the TPR*-tree [35], but they are out of the scope of this thesis.

2.3.1 Octree

An octree is a data structure to generate partitions of a 3D space. Each node has exactly eight children, except the leaves. There are two types of octree: point region (PR) octree and matrix-based (MX) octree. In a PR octree, the node represents one three-dimensional point, which is the centre of the subdivision, and it is one of the corners of the eight children. In an MX octree, the division point is implicitly the centre of the space. The root node of a PR octree represents infinite space, while in an MX octree, it represents a finite bounded space. When can see an example in the figure 2.3.1.

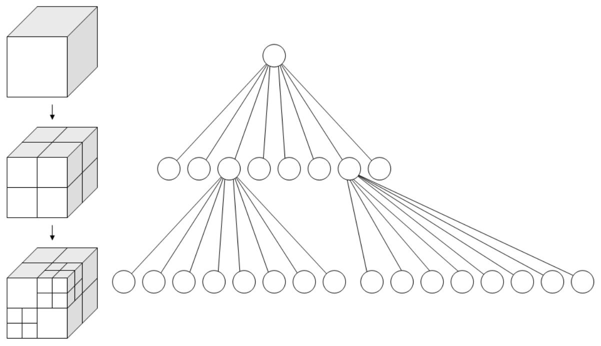

Figure 2.3.1: Left: Recursive subdivision of a cube into octants. Right: The corresponding octree. [28]

This data structure was proposed by Donald Meagher in 1980 [26] and the construction goes as follows. First, it divides the current 3D volume into eight boxes. If any box has more than one point then apply again the algorithm. If the box has zero or one point, stop building that branch.

Quadtree

A quadtree is another tree data structure proposed earlier by Raphael Finkel and Joun Louis Bentley in 1974 [10]. This data structure is the two-dimensional version of the Octree. Similar to the Octree, the algorithm divides the 2D space into four quadrants recursively. No more detail is needed for this data structure because our data is 3D-based and this solution works for 2D points.

2.3.2 R-tree

An R-tree is a data structure used for spatial access methods for k-dimensional data. Antonin Guttman proposed the R-tree in 1984 [13] and the key idea is to group nearby points under a tight bounding box. The construction of the tree is bottom-up, which means the leaves are constructed first and it starts associating nodes with their bounding boxes until the root is reached. One visualization of the space partition made by an R-tree in a three-dimensional space can be seen in the figure 2.3.2

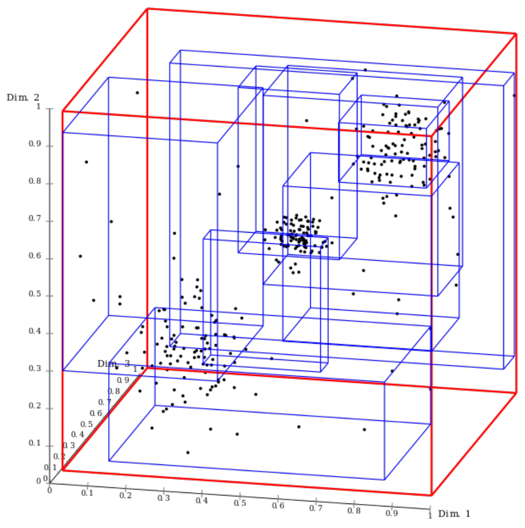

Figure 2.3.2: Visualization of an R*-tree for 3D points [9]

This tree has different variations like priority R-tree, R*-tree, R$^+$ tree, RR*-tree, Hilbert R-tree and X-tree. One common variation used in the touching point problem is R*-tree. This variant has a slightly higher construction cost, but on the other hand, it will have a lower cost while querying. The main difference between R*-tree and R-tree is that R*-tree reduces the overlapping of subtrees, pruning more branches while querying and improving the performance, therefore.

2.3.3 *k*-d trees

k-d tree is a data structure for space-partitioning *k*-dimensional points. It is a characteristic case of space partitioning trees called binary space partitioning (BSP) trees. Being a BSP tree means that each time it splits the space it is done in two convex subsets. It was proposed in 1975 by Jon Louis Bentley [5]. The construction of the *k*-d tree is done recursively. First of all, one point and one axis are selected. Originally, the dimension was chosen by round-robin and the point was chosen by the median in that axis, but in this thesis, we will explore some other options. The axis and the point will form a hyperplane aligned with the axis. Then, the algorithm splits the points into two subsets: the point at the "left" of the plane and the points at the "right". Finally, the left and right subtrees are constructed using the same algorithm. One example of the execution of this algorithm can be seen in the figure 2.3.3.

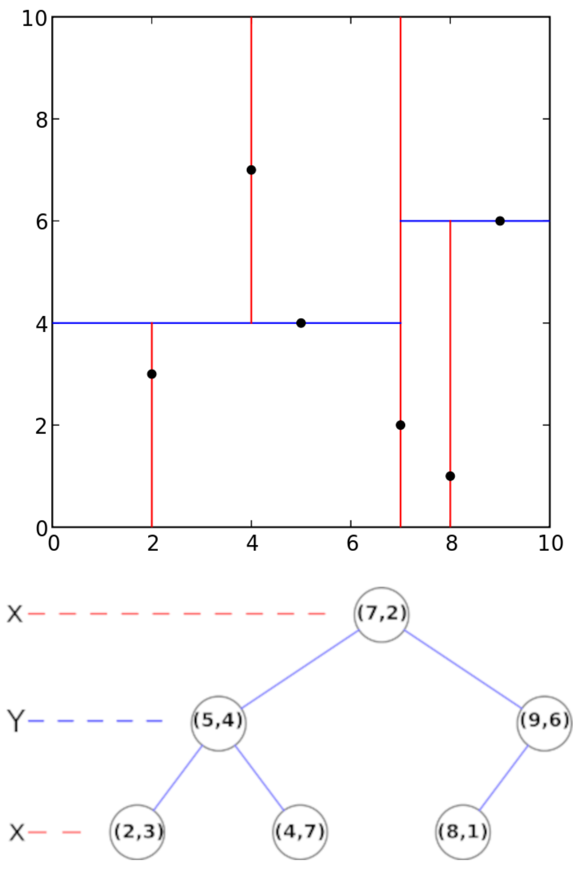

Figure 2.3.3: k-d tree decomposition for a point set [20]

2.4 Previous research

There are several studies on spatial-partitioning structures, but a little few focus on the touch detection problem. In their thesis, Adamsson and Vorkapic [1] analyze the performance of different spatial-partitioning structures: *k*-d trees, vantage-point trees and octrees. They discovered that vantage-point trees are the best option with small populations, but *k*-d trees work better with larger networks. Because of their result, we choose *k*-d trees over other data structures. In Beredent and Brask's thesis [7], they analyze the memory efficiency for R-trees and R*-trees. They analyzed the performance with realistic neuron densities and found out that R*-tree has the same good scalability properties as k-d trees.

Compared with the studies on spatial-partitioning structures on the touch detection problem, more studies are trying to optimize the k-d trees partition for different applications, such as computer graphics or curve analysis. Hu, Nooshabadi and Ahmadi in their paper [16] propose a method for constructing k-d trees and doing nearest neighbor search (NNS) with a speed up of a factor of 30, compared with a serial CPU. Wald and Havran [37] propose an heuristic for choosing the dimension and splitting point called surface area heuristic that will speed up for raytracing with objects based on triangles. In other paper by different authors [23], they propose a heuristic called Curve Complexity Heuristic, which aim is to allow the exploration of 3D curves based on the neighbourhood.

Chapter 3

Methods

3.1 Reconstruction of a neuronal network

First of all, we chose to reconstruct the neuronal network based on a single neuron, a fast-spiking interneuron. NeuroMorpho.Org [3, 4] is an archive containing multiple reconstructions of different types of neurons from across different researches. The choice was completely arbitrary and was made to simplify the biological complexity of the problem while keeping the computational complexity because fast-spiking interneurons allow the creation of any kind of synapse. The exact neuron was "NMO_36576" from one research of 2013 [32], the figure 3.1.1 contains a representation of this neuron. This specimen belonged to the subiculum of a rat and one characteristic of note is that the neuron spatial distribution is not uniform, $298.75\mu m$ x $538.69\mu m$ x $66.28\mu m$.

To avoid reading the SWC file every time that it was needed to replicate the neuron, it was designed another file called RPL. This new format file defined a neuron per line and each line contained which transformations were going to be applied. The possible transformations defined were rotation and translation. Scaling and shearing were discarded because they deform the morphological definition of the neuron. A transformation was defined by three values: the first value defined which transformation (0 for translation and 1 for rotation), the second value defined which axis (0 means axis x, 1 means axis y, 2 means axis 3) and the third value defined the amount of transformation in micrometres for translation and radians for rotation. Any amount of transformation could be concatenated, so every possible neuron was possible.

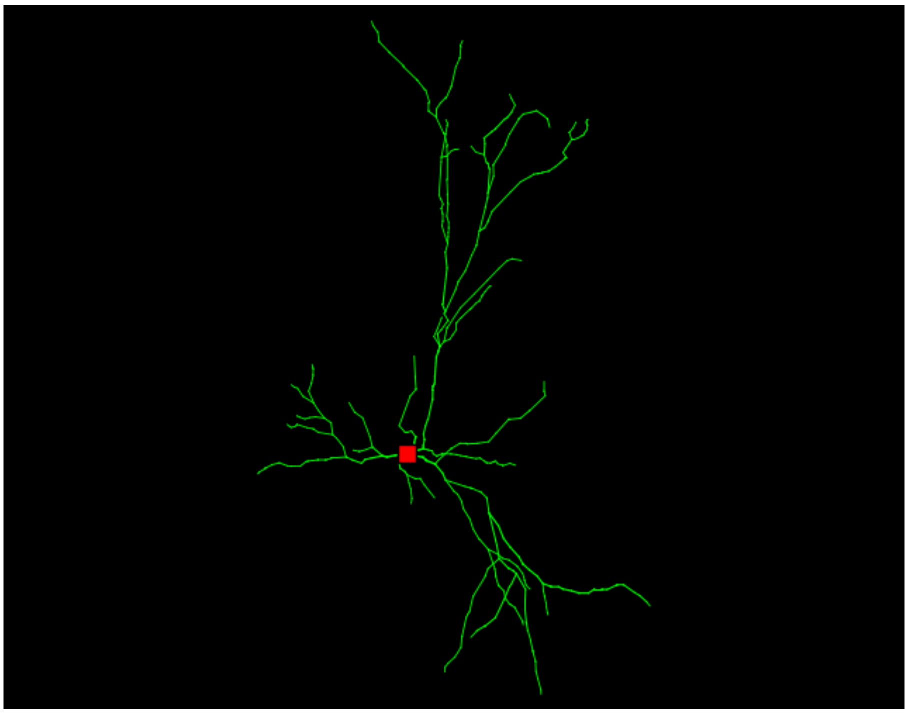

Figure 3.1.1: Fast-spiking interneuron used in the thesis.

3.2 Data structure and construction algorithm

3.2.1 *k*-d tree

When building a *k*-d tree are necessary a list of the points and a depth. As shown in algorithm 1, if the list of nodes is empty, we have finished building the tree and it is none. Otherwise, it first needs to find the dimension along which the splitting plane (or hyperplane) is going to be defined and then it chooses a point to create the plane (or hyperplane). Then, the algorithm divides the space into two subsets that are at the left or the right of the plane (or hyperplane) along that dimension. Finally, it creates the subtrees recursively increasing the depth by one. The returned node needs to contain not only the point and the left and right subtree but also the splitting dimension, so when performing a spatial-query, it is possible to know which dimension is responsible for the branches.

Algorithm 1 Constructs a k-d tree

```
Require: $N$ is the list of points to construct the $k$-d tree with.
Require: $k$ is the depth of the $k$-d tree.

 1: function BuildTree($N, k$)
 2:     if $|N| = 0$ then
 3:         return nullptr
 4:     end if
 5:     $d$, $p \leftarrow$ SplittingFunction($N, k$)   $\triangleright$ $d$ is the splitting dimension and $p$ is the point
 6:     LeftSplit $\leftarrow \{n \in N \mid n_d \leq p_d \wedge n \neq p\}$
 7:     RightSplit $\leftarrow \{p \in N \mid p_d < n_d \wedge n \neq p\}$
 8:     LeftChild $\leftarrow$ BuildTree($LeftSplit, k + 1$)
 9:     RightChild $\leftarrow$ BuildTree($RightSplit, k + 1$)
10:     return new Node($p$,$d$,LeftChild,RightChild)
11: end function
```

3.3 Splitting function and heuristics

The canonical and original method for dividing the list of nodes is really simple and still leaves a lot of room for improvement. That is the reason because there are several studies trying to improve the splitting function for different applications. This thesis studied how the proposed splitting functions work for the touching point detection.

3.3.1 Canonical splitting function

The canonical method was first described with, the k-d tree in its original paper [5]. This method selects the splitting dimension by round-robin, that means the dimension cycles along all the axis. For example, in a 3D tree, the first plane will be aligned with the x-axis, the root's children will both have y-aligned planes, the grandchildren's planes will be aligned with the z-axis, the root's great-grandchildren will all have x-aligned planes, and so on. Once the axis is chosen, the algorithm chooses the median point in that axis from all the points. One possible implementation of this algorithm is the algorithm 2.

The main advantage of this method is that it leads to a balanced k-d tree, that means that each leaf of the tree is approximately at the same distance to the root. Selecting the median is a complex task were different approaches can be done. Sorting all the points with a sorting algorithm such as heapsort or mergesort, will lead into a complexity $\mathcal{O}(n \log n)$. An improved solution with complexity $\mathcal{O}(n)$ called PICK [6] can be also implemented. A common implementation that is approximate selects random points from

Algorithm 2 Canonical splitting function

```
Require: $N$ is the list of points to construct the $k$-d tree with.
Require: $k$ is the depth of the $k$-d tree.
Require: $K$ is the number of dimensions.

1: function SplittingFunction($N, k$)
2:     axis $\leftarrow$ $\pmod K$
3:     select median by axis from N
4:     return (axis,median)
5: end function
```

the total of points and use the median of those points. Although it is not optimal and does not ensure a balanced tree, it is widely used and often results in nicely balanced trees. In this thesis, we have used a built-in function from C++11 called "std::nth_element". It is a partial sorting algorithm that rearranges the elements in the vector such as the element pointed at by the n^{th} is changed so it would occur in that position if the vector was sorted, all elements before are less than or equal to the new n^{th} element and the complexity is linear to the number of elements $\mathcal{O}(n)$ [18].

3.3.2 Surface area heuristic

The surface area heuristic (SAH) was proposed by MacDonald and Booth in their paper of 1990 [24]. The main idea behind their proposal is that the number of rays likely to intersect a convex object is roughly proportional to its surface area, assuming that the ray origins and directions are uniformly distributed throughout the space. We believed there was a similarity with our task because our neurons are uniformly distributed in the space, and the orientation of the touching point does not matter, so it can be said that the "ray direction" is also evenly distributed. Instead of using the straight-forward implementation from the original description of the algorithm that is $\mathcal{O}(n^2)$ for building the tree, we have used other implementation [37] that is $\mathcal{O}(n \log^2 n)$. This implementation is the algorithm 3.

The called function in line 27, SAH, basically is defined as:

$$SAH(p, N, N_l, N_r, N_p) = \lambda * (K_T + min(\frac{(N_l + N_p)|\{t \in N \mid t_i < p_i\}| + N_r|\{t \in N \mid t_i > p_i\}|}{|N|}, \frac{N_l|\{t \in N \mid t_i < p_i\}| + (N_p + N_r)|\{t \in N \mid t_i > p_i\}|}{|N|}))$$

There are two values in this function that are unknown λ and K_T, λ is a factor to

Algorithm 3 Surface area heuristic

Require: N is the list of points to construct the k-d tree with.
Require: k is the depth of the k-d tree.
Require: K is the number of dimensions.

```
 1: function SplittingFunction($N, k$)
 2:     $\hat{C}, \hat{d}, \hat{p} \leftarrow (\infty, \infty, \emptyset)$
 3:     for $i = 1..K$ do
 4:         $E \leftarrow \emptyset$
 5:         for all $p \in N$ do
 6:             $E \leftarrow E \cup (p, p_i, +) \cup (p, p_i, -)$
 7:         end for
 8:         sort($E$) ▷ First, it orders by $p_i$ and if they are equal, then it goes first the $-$
    and then the $+$
 9:         $N_l, N_p, N_r \leftarrow (0, 0, |N|)$
10:         for $j = 1..|E|$ do
11:             $p \leftarrow E_{j,p}$
12:             $p^+, p^-, p^| \leftarrow 0$
13:             while $j < |E| \wedge E_{j,\xi} = p_\xi \wedge E_{j,type} = -$ do
14:                 $p^- \leftarrow p^- + 1$
15:                 $j \leftarrow j + 1$
16:             end while
17:             while $j < |E| \wedge E_{j,\xi} = p_\xi \wedge E_{j,type} = |$ do
18:                 $p^| \leftarrow p^| + 1$
19:                 $j \leftarrow j + 1$
20:             end while
21:             while $j < |E| \wedge E_{j,\xi} = p_\xi \wedge E_{j,type} = +$ do
22:                 $p^+ \leftarrow p^+ + 1$
23:                 $j \leftarrow j + 1$
24:             end while
25:             $N_p \leftarrow p^|$
26:             $N_r \leftarrow N_r - p^| - p^-$
27:             $C \leftarrow SAH(p, N_l, N_r, N_p)$
28:             if $C < \hat{C}$ then
29:                 $\hat{C}, \hat{d}, \hat{p} \leftarrow (C, i, p)$
30:             end if
31:             $N_l \leftarrow N_l + p^+ + p^|$
32:             $N_p \leftarrow 0$
33:         end for
34:     end for
35:     return ($\hat{d}$, $\hat{p}$)
36: end function
```

prioritize cutting empty space, and it will be 0.8 if N_l or N_r is 0, otherwise it is 1. K_T is a constant that represents the cost of a transversal step in the tree. We established it to 0.2 based on other paper [23]. The implemented version is not exactly the proposed in the original article. In line 6, they firstly clamp the triangle containing the point and in case the resulting triangle is planar, then it changes what is added to E, by $E \cup (p, p'_{min,k}, |)$. We have done this because we are creating one k-d tree per neuron, so the neuron will always be inside the voxel of the k-d tree.

3.3.3 Curve complexity heuristic

The curve complexity heuristic propose a new splitting function based on SAH to optimize k-d trees for curves abstractions [23]. The main idea behind their proposal is to adapt SAH so they can perform radius nearest curve search better. We believed there is a great similarity between their proposal and our problem because we also want to find all the nearest neurons given a query point **q** and a distance **r**. A possible implementation could be the algorithm 4.

Algorithm 4 Curve complexity heuristic

```
Require: $N$ is the list of points to construct the $k$-d tree with.
Require: $k$ is the depth of the $k$-d tree.
Require: $K$ is the number of dimensions.

 1: function SplittingFunction($N, k$)
 2:     $\hat{C} \leftarrow \infty$
 3:     $\hat{p} \leftarrow \emptyset$
 4:     $\hat{d} \leftarrow \emptyset$
 5:     for i = 1..K do
 6:         $C_i \leftarrow 0$        ▷ $C(T) = C_{\text{traversal}}(T) + C_{\text{backtrack}}(T) - nT_{\text{dist}}$
 7:         select median by axis from N
                  ▷ $C_{\text{trasversal}}(T) = T_{\text{trasversal}} + [P(T_l|T)l + P(T_r|T)r]T_{\text{dist}}$
 8:         $C_i \leftarrow 0.2 + l * \frac{\text{Volume}(\{p \in N \mid p_i < \text{median}_i\})}{\text{Volume}(N)} + r * \frac{\text{Volume}(\{p \in N \mid p_i > \text{median}_i\})}{\text{Volume}(N)}$
                  ▷ $C_{\text{backtracl}}(T) = \lambda(\log_{\frac{1}{\rho}}(n) + \log_{\frac{1}{\tau}}(n))/2$        ▷ $\rho := \frac{l}{n}$, $\tau := \frac{r}{n}$
 9:         $C_i \leftarrow C_i + \lambda * (\frac{\log(n)}{\log(\frac{|\{p \in N \mid p_i < \text{median}_i\}|}{|N|})} + \frac{\log(n)}{\log(\frac{|\{p \in N \mid p_i > \text{median}_i\}|}{|N|})})$
10:         if $C_i < \hat{C}$ then
11:             $\hat{C} \leftarrow C_i$
12:             $\hat{p} \leftarrow$ median
13:             $\hat{d} \leftarrow i$
14:         end if
15:     end for
16:     return ($\hat{d}$, $\hat{p}$)
17: end function
```

In their paper, they do not study the time complexity of their solution, so we are going to do it. Assuming that the volume functions and selecting the median are $\mathcal{O}(n)$ and the other operations are $\mathcal{O}(1)$. To obtain the cost function, it will sum the cost of finding the median (n), the find of the volumes function (n), the cost of splitting $(2n)$ and the cost of constructing the subtrees recursively $(2T(\frac{n}{2}))$:

$$\begin{aligned}
T(n) &= n + n + 2n + 2T(\frac{n}{2}) \\
&= 4n + 2T(\frac{n}{2}) \\
&= 4n + 2[4\frac{n}{2} + 2T(\frac{n}{4})] = 8n + 4T(\frac{n}{4}) \\
&= 8n + 4[4\frac{n}{4} + 2T(\frac{n}{8})] = 12n + 8T(\frac{n}{8}) \\
&= 12n + 8[4\frac{n}{8} + 2T[\frac{n}{16}]] = 16n + 16T(\frac{n}{8}) \\
&= \cdots = 4 * i * n + 2^i T(\frac{n}{2^i}), i \in \mathbb{Z}^+ \\
&\text{This ends when } \frac{n}{2^i} = 1, \text{ because } T(1) \in \mathcal{O}(1) \\
&n = 2^i \implies i = \log{(n)} \\
&\text{Therefore, if we substitute in the last one, we get} \\
T(n) &= 4n\log{(n)} + 2^{\log{(n)}} = 4n\log{(n)} + n \in \mathcal{O}(n\log{(n)})
\end{aligned}$$

3.3.4 Median of the hyperplane with maximum variance

This heuristic is our own proposal and is similar to curve complexity heuristic, because it chooses the splitting point by the median and our heuristic defines the splitting dimension. The main idea behind our proposal is that the dimension with a higher variance means that the points are more separated between them and therefore if we split, we are improving. To avoid a costly implementation of the variance, we have implemented the algorithm proposed by Donald Knuth in his book "The art of computer programming. Vol. 2" [21] (page 232). So, the algorithm implemented can be seen in the algorithm 5.

The time complexity of this function is trivially $\mathcal{O}(n)$, and therefore when added to the building tree function it is also $\mathcal{O}(n\log{(n)})$ as the other heuristics.

Algorithm 5 Median of the hyperplane with maximum variance

Require: N is the list of points to construct the k-d tree with.
Require: k is the depth of the k-d tree.
Require: K is the number of dimensions.

```
1: function SplittingFunction($N, k$)
2:     $n \leftarrow 0$
3:     $M \leftarrow \{0, 0, 0\}$
4:     $S \leftarrow \{0, 0, 0\}$
5:     for all $p \in N$ do
6:         $n \leftarrow n + 1$
7:         for i = 1..K do
8:             $\delta \leftarrow p_i - M_i$
9:             $M_i \leftarrow M_i + \frac{\delta}{n}$
10:            $S_i \leftarrow S_i + \delta * (p_i - M_i)$
11:        end for
12:    end for
13:    $d \leftarrow$ MAX_ELEMENT($S$) ▷ max_element returns the index of the max element in S
14:    select median by $d$ from $N$
15:    return (d,p)
16: end function
```

3.3.5 Minimum variance union

This heuristic is also our own proposal and it is an improvement from the previous. It also uses the variance of the points in that dimension as a guide, but in this case it tries to minimize the void. To do so, it will minimize for all the points the sum of the variances at the left subset and the right subset. To avoid go through all the points twice, it calculates all the variance at the same time using memoization. This technique consists on save some values so you do not calculate them again. We are exploiting it because if the data points are ordered, then the median of the left subset for the i-th data point is the median of the $i-1$-th data point but with some update, also happens for the right subset. In this case the i-th median is an updated version of the $i+1$-th median. The purposed algorithm works as follow 6.

The time complexity of this function is trivially $\mathcal{O}(n \log(n))$ because there is a sort function, and therefore when added to the building tree function it is also $\mathcal{O}(n \log^2(n))$ as the surface area heuristic.

Algorithm 6 Minimum variance union

```
Require: $N$ is the list of points to construct the $k$-d tree with.
Require: $k$ is the depth of the $k$-d tree.
Require: $K$ is the number of dimensions.

1: function SplittingFunction($N, k$)
2:     $\hat{S} \leftarrow \infty$
3:     $\hat{p} \leftarrow \emptyset$
4:     $\hat{d} \leftarrow \emptyset$
5:     for i = 1..K do
6:         sort_by_dimension(N,i)
7:         $S_l \leftarrow$ Array of size $|N|$ with 0
8:         $S_r \leftarrow$ Array of size $|N|$ with 0
9:         $M_l \leftarrow$ Array of size $|N|$ with 0
10:        $M_r \leftarrow$ Array of size $|N|$ with 0

11:        for $j \in [1, |N|)$ do
12:            $\delta \leftarrow N_{j,i} - M_{l,j-1}$
13:            $M_{l,j} \leftarrow \delta/(i+1)$
14:            $S_{l,j} \leftarrow \delta * N_{j,i} - M_{l,j}$          ▷ Compute the variance left to right

15:            $\delta \leftarrow N_{|N|-j-1,i} - M_{l,|N|-j}$
16:            $M_{r,|N|-j-1} \leftarrow \delta/(i+1)$
17:            $S_{r,j} \leftarrow \delta * N_{|N|-j-1,i} - M_{r,|N|-j-1}$          ▷ Compute the variance right to left
18:        end for
19:        for $j \in [0, |N|)$ do
20:            if $S_{l,j} + S_{r,j} < \hat{S}$ then
21:                $\hat{S} \leftarrow S_{l,j} + S_{r,j}$
22:                $\hat{p} \leftarrow N_j$
23:                $\hat{d} \leftarrow i$
24:            end if
25:        end for
26:    end for
27:    return ($\hat{d}$, $\hat{p}$)
28: end function
```

3.4 Neuron touch point task

Due to the fact that the implemented solution uses one k-d tree per neuron, the neuron touch point task is trivial to implement, because it only needs to look for if there is a point which distance to the query point is less than a certain threshold. In our case, we choose that threshold to be $3\mu m$, but it is a variable of our search function (algorithm 7).

On a sight, it is trivial to see that the computational cost in the worst case is $\mathcal{O}(n)$,

Algorithm 7 Touchpoint query

Require: root is a k-d tree, root_p is the point represented in the node, root_d is the splitting dimension, root_l, root_r are the left and right children.
Require: q is the query point where the touching point is looked for.
Require: dist is the distance to define a touching point.

```
1: function FindTouchpoint(root, $q$, dist)
2:     if root = nullptr then
3:         return nullptr
4:     end if
5:     $\delta \leftarrow |\text{root}_p - q|$
6:     if $\delta \leq$ dist then
7:         return root
8:     else
9:         if $q_{\text{root}_d} \leq \text{root}_{\text{root}_d}$ then    ▷ $\text{root}_{\text{root}_d}$ means the value of the root in the dimension which was chosen as splitting dimension
10:            res ← FindTouchpoint($\text{root}_l$, $q$, dist)
11:            if res ≠ nullptr then
12:                return res
13:            end if
14:            res ← FindTouchpoint($\text{root}_r$, $q$, dist)
15:            if res ≠ nullptr then
16:                return res
17:            end if
18:        else
19:            res ← FindTouchpoint($\text{root}_r$, $q$, dist)
20:            if res ≠ nullptr then
21:                return res
22:            end if
23:            res ← FindTouchpoint($\text{root}_l$, $q$, dist)
24:            if res ≠ nullptr then
25:                return res
26:            end if
27:        end if
28:        return nullptr
29:    end if
30: end function
```

but that probably in average will be $\mathcal{O}(\log(n))$ because every time that the algorithm branches, it can discard the half of the data points.

3.5 Task parallelization

OpenMP 5.2 was used to introduce parallel programming. OpenMP is an API to implement shared-memory multiprocessing programming in C, C++ and Fortran. To avoid any race condition and to maximize the parallelization of the processes, it was only applied when there were no shared resources between the possible sub-processes. It was widely used for creating the subtrees, where every time that it was needed to call recursively the building function with a subset of the data points, it was done with parallel tasks. Also, when performing the query makes use of parallelizing the search of all the k-d trees.

3.6 Test cases

Firstly, the distance to define a touchpoint was fixed to $3\mu m$. Although it could seem to be arbitrary, it was the middle value between $0.5\mu m$ and $5\mu m$, the range proposed by other studies [15, 31]. Once the distance to define a touchpoint was fixed, there were two possible variables to vary for benchmarking the scalability of the data structure and heuristics. The variable for each test case is the number of neurons of the neuronal network given the density and the neuron density of the search space given the number of neurons.

In order to reach realistic values for the number of neurons, we fixed the density to 16991 neurons/mm^3 [36], but there could be other densities such us 2048neurons/mm^3 and 2790neurons/mm^3 [19] according to our neuron that is a fast-spiking interneuron in the subiculum. The number of neurons was a range between 150 neurons and 85000 neurons. This number is lower than the total amount of any type of neurons in the subiculum of a rat's brain which is between 46000 and 330000 neurons [27].

For the other test case, the number of neurons was fixed to 25230 neurons, so it does not have a high time cost. Then the density of neurons by mm^3 was defined in a range of 50 neurons/mm^3 and 20000 neurons/mm^3. The value of the upper bound was chosen to keep realistic densities [19]. Another test case was derivated with higher densities, between 50000 neurons/mm^3 and 500000 neurons/mm^3, to be able to use the results to other types of neurons and regions of the brain.

For the purpose of creating the tests of every test case, we developed a script (algorithm 8) that given a density and the number of neurons, it will rotate randomly each neuron and then translate it into the space so the distance of the origin of all the neurons is in a grid such as the distance between them maintains the density. This system uses the RPL files explained in section 3.1.

Algorithm 8 Create a test

```
Require: $N$ is the number of neurons to use in the test
Require: $D$ is the density in neurons/$mm^3$ to use in the test
1: procedure CreateTest($N, D$)
2:     $\delta \leftarrow 1000 * \sqrt[3]{\frac{1}{D}}$
3:     $n_1 \leftarrow \lfloor \sqrt[3]{N} \rceil$
4:     $n_2 \leftarrow \lfloor \sqrt{\frac{N}{n_1}} \rceil$
5:     $n_3 \leftarrow \lfloor \frac{N}{n_1 n_2} \rceil$
6:     for i = 1..$n_1$ do
7:         for j = 1..$n_2$ do
8:             for k = 1..$n_3$ do
9:                 rotation ← RndChoice([[], [0], [1], [2], [0,1], [0,2], [1,2], [0,1,2]])
10:                rotation ← RndShuffle(rotation)
11:                for all $r \in$ rotation do
12:                    $\phi \leftarrow$ RndUniform($-\pi$, $\pi$)
13:                    Write('{} {} {} '.format(1, $r$, $\phi$))
14:                end for
15:                Write('0 0 {} 0 1 {} 0 2 {}\n'.format($\delta * i$, $\delta * j$, $\delta * k$))
16:            end for
17:        end for
18:    end for
19: end procedure
```

3.7 Computer specification

During this project, we used a personal computer running Debian Live 11.3.0 Standard with an architecture Amd64. The processor is an AMD FX(tm)-8350 with eight cores, 8 threads, working at 4.0 GHz, 384KB of caché L1, 8MB of caché L2 and 8MB of caché L3. Furthermore, it has 16 GB of DDR3 RAM at 1600 MHz distributed in 3 modules: one module of 8 GB (Kingston KHX1600C9D3K2/8GX) and two modules of 4 GB (Kingston 99U5584-012.A00LF). We hadn't physical access to the computer, so it runs an OpenSSH server.

Chapter 4

Results

The results presented were obtained from 5 individual runs for each test case to avoid outliers bias our results. The ranges for each test case (section 3.6) were sampled with 200 points each, except the range to extrapolate the results to other cases that was sampled with 450 points. The mean performance of each heuristic is shown in milliseconds.

4.1 Neuron Density

4.1.1 Computation time

Benchmarks for different densities were made in the range 50-85000 neurons/mm^3 with different heuristics. The figure 4.1.1 shows in the left graph the average building time of the k-d tree, the middle graph shows its performance when doing the query and the right graph show how many touchpoints were detected. Due to memory usage for surface area heuristic and time consumption for minimum variance union, they were not out of the usage limits.

4.1.2 Average approximation

Using the previous benchmark, a fitting to two different functions, $m * n + c \in \mathcal{O}(n)$ and $m * log(n) + c \in \mathcal{O}(log(n))$, were made. The figure 4.1.2 shows how it fits for every heuristic and the table 4.1.1 shows the values of the fitted parameters and the coefficient of determination, R^2.

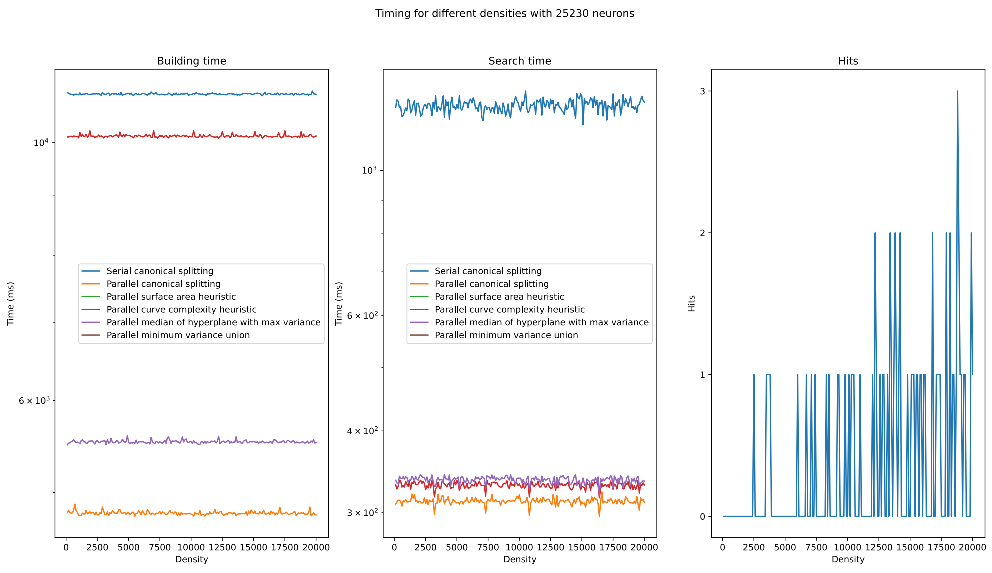

Figure 4.1.1: Performance when varying density

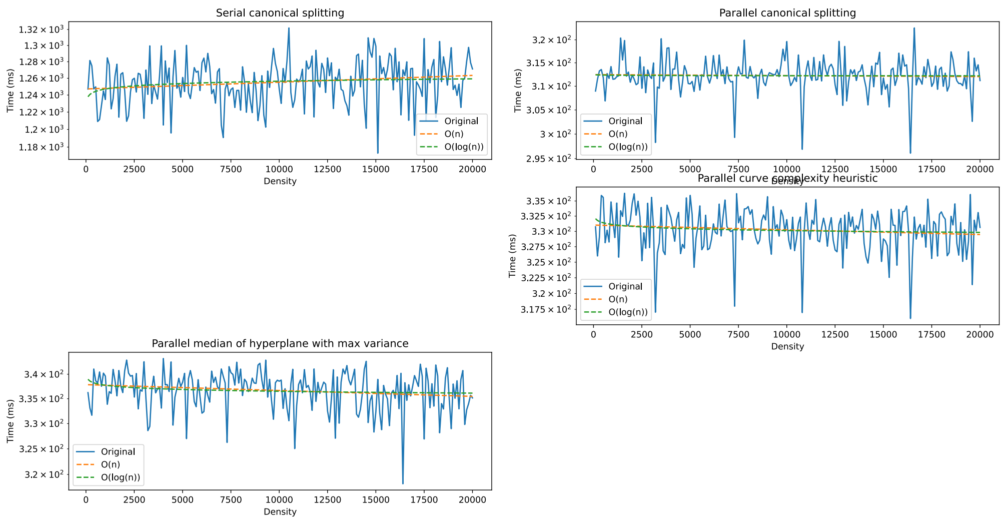

Figure 4.1.2: Approximation to different curves for each benchmark

	m*n+c			m*log(n)+c		
Heuristic	Slope	Bias	R^2	Slope	Bias	R^2
Serial canonical splitting	0.00081	1247.19350	0.03084	4.05824	1219.13713	0.02088
Parallel canonical splitting	-0.00002	312.46810	0.00121	-0.04199	312.62670	0.00012
Parallel curve complexity heuristic	-0.00008	331.03052	0.01588	-0.42717	334.06362	0.01307
Parallel median of hyperplane with max variance	-0.00012	337.86543	0.02981	-0.53776	341.46615	0.01643

Table 4.1.1: Approximation of the query time when varying density for different heuristic to some functions

4.2 Amount of neurons

4.2.1 Computation time

Benchmarks for different amount of neurons were made in the range 150-20000 neurons/mm^3 with different heuristics. The figure 4.2.1 shows in the left graph the average building time of the k-d tree, the middle graph shows its performance when doing the query and the right graph show how many touchpoints were detected.

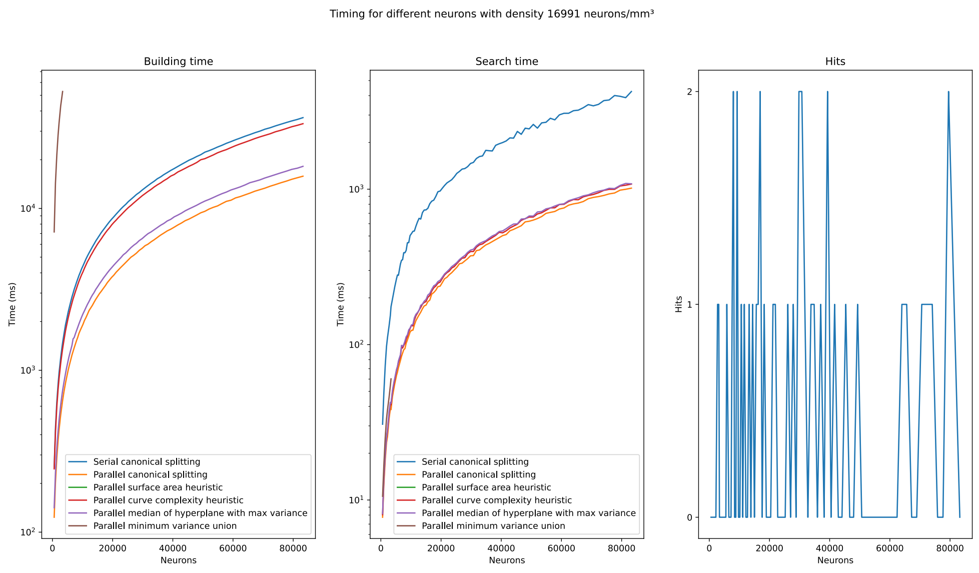

Figure 4.2.1: Performance when varying amount of neurons

4.2.2 Average approximation

Using the previous benchmark, a fitting to two different functions, $m * n + c \in \mathcal{O}(n)$ and $m * log(n) + c \in \mathcal{O}(log(n))$, were made. The figure 4.2.2 shows how it fits for every heuristic and the table 4.2.1 shows the values of the fitted parameters and the coefficient of determination, R^2.

	m*n+c			m*log(n)+c		
Heuristic	Slope	Bias	R^2	Slope	Bias	R^2
Serial canonical splitting	0.04975	-1.93794	0.99865	936.47290	-7738.1621	0.77479
Parallel canonical splitting	0.01246	-1.29551	0.99958	235.04158	-1943.93362	0.77944
Parallel curve complexity heuristic	0.01319	-0.75093	0.99956	248.93553	-2058.14906	0.77913
Parallel median of hyperplane with max variance	0.01337	0.85753	0.99955	252.65924	-2087.86438	0.78123
Parallel minimum variance union	0.01741	-0.09999	0.99699	27.746100	-172.11974	0.92816

Table 4.2.1: Approximation of the query time when varying number of neurons for different heuristic to some functions

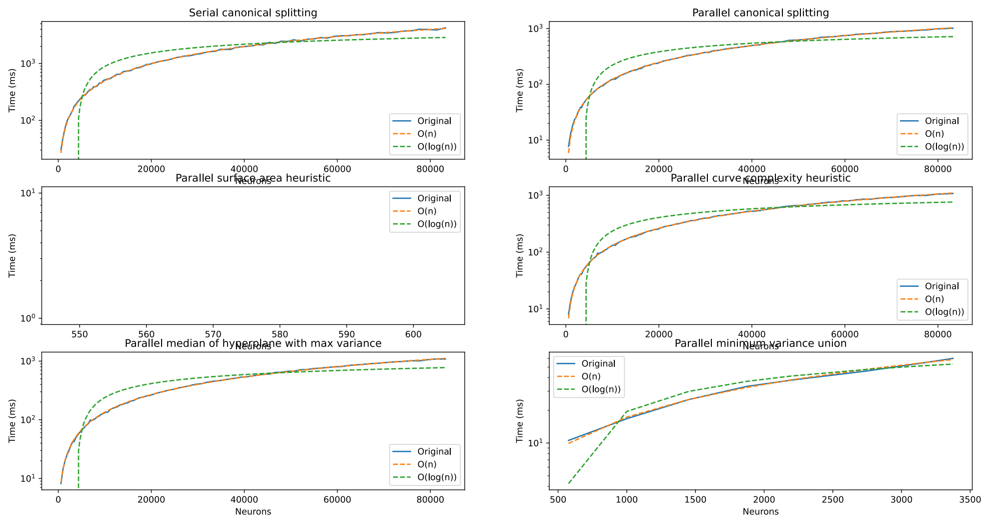

Figure 4.2.2: Approximation to different curves for each benchmark

4.3 High density

4.3.1 Computation time

Benchmarks for different densities out of realistic values were made in the range 150-20000 neurons/mm^3 for different heuristics to try to extrapolate the results to other types of neuronal networks. The figure 4.3.1 shows in the left graph the average building time of the k-d tree, the middle graph shows its performance when doing the query and the right graph show how many touchpoints were detected.

4.3.2 Average approximation

Using the previous benchmark, a fitting to two different functions, $m * n + c \in \mathcal{O}(n)$ and $m * log(n) + c \in \mathcal{O}(log(n))$, were made. The figure 4.3.2 shows how it fits for every heuristic and the table 4.3.1 shows the values of the fitted parameters and the coefficient of determination, R^2. Due to memory usage for surface area heuristic and time consumption for minimum variance union, they were not out of the usage limits.

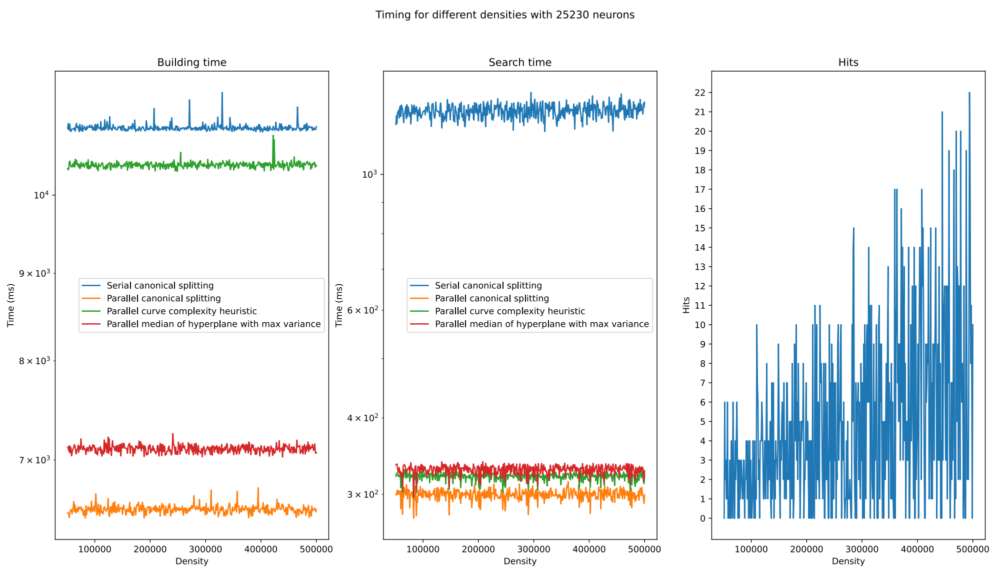

Figure 4.3.1: Performance when extrapolating results with density

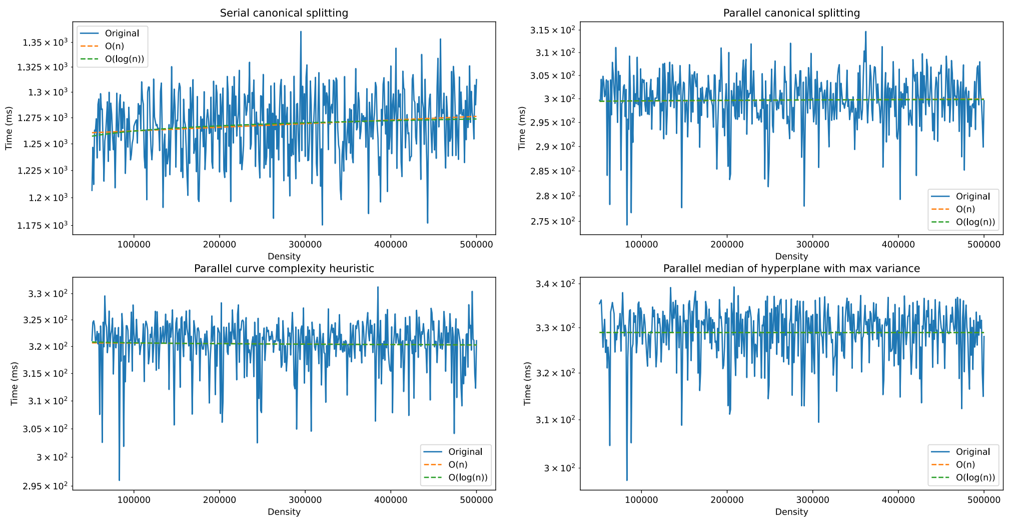

Figure 4.3.2: Approximation to different curves for each benchmark

	m*n+c			m*log(n)+c		
Heuristic	Slope	Bias	R^2	Slope	Bias	R^2
Serial canonical splitting	0.00003	1258.86561	0.02032	7.23026	1178.97566	0.01771
Parallel canonical splitting	0.00000	299.42317	0.00056	0.16195	297.71509	0.00026
Parallel curve complexity heuristic	-0.00000	320.71691	0.00057	-0.23793	323.42042	0.00085
Parallel median of hyperplane with max variance	-0.00000	328.95618	0.00001	-0.00972	329.04446	0.00000

Table 4.3.1: Approximation of the query time when extrapolating the density for different heuristic to some functions

Chapter 5

Discussion

We investigated how performance differ for different heuristics in a *k*-d tree. The results show that there is not a significant difference in performance between canonical, SAH, CCH, median of hyperplane with max variance and minimum variance union heuristics.

5.1 Building performance

Firstly, we needed to analyze how each heuristic affect to the time of building the *k*-d tree. We considered that if an heuristic takes more time to build in parallel than the canonical in serial it is not worth it, due to the great amount of time. In figures 4.1.1, 4.2.1 and 4.3.1, we can observe the difference between the building time for the different heuristics in the left side. In those figures, we can observe that obviously implementing everything with parallelization has sped up almost by a factor of 3.

The figure 4.2.1 shows that the minimum variance union heuristic includes a lot of overhead and therefore it is not worth it. Similar happens with surface area heuristic. Although a $\mathcal{O}(n \log^2(n)$ has been implemented, the memory usage was so high than anything above 150 neurons were over 16 GiB and therefore this implementation was unfeasible. This is because in line 4 to 7 of algorithm 3 the number of data saved is 3 times the size of N, so it will require a lot of memory. In the other hand, curve complexity heuristic and median of hyperplane with maximum variance have a lower cost than the serial canonical splitting what means that they could be feasible to use in a real case.

5.2 Querying performance

Once the tree has been built, we were able to query it looking for all touchpoints close to a given point in our reconstruction of the neuronal network. We have observed in figures 4.1.1 and 4.2.1 (middle figure) that all the heuristics performed the same . There were no significant differences in the performance. We firstly believed it is since the points are shown sparse that the heuristics do not have any effect, the number of touchpoints can be observed in the right figures in 4.1.1 and 4.2.1. To prove this fact, we repeated the experiments increasing the density. The results in the figure 4.3.1 show that although the amount of touchpoints is higher, there is no better performance of the k-d trees built with the heuristics.

Theoretically, we know that the worst-case for querying is $\mathcal{O}(n)$, but on average it should be $\mathcal{O}(log(n))$. Experimentally, we saw that our results fit better to a linear cost rather than a logarithmic cost. Tables 4.1.1, 4.2.1 and 4.3.1 shows that the lineal function fits with a lower R^2 (coefficient of determination) and it is also evident in the figures 4.1.2, 4.2.2 and 4.3.2. Analysing the reasons for the lack of improvement in the performance, we ended up with three possible reasons. The implemented algorithm to perform the touchpoint query (algorithm 7) could be improved if in lines 14 to 17 and 23 to 26 another condition is added where those lines are only executed if the distance in that dimension between the root and the query point is less or equal to the threshold distance. Another reason could be if the distance between the query point and the data is far, the number of points examined could be upper bounded by 2^n [11]. This led to what we believed to be the main reason. At the beginning of this thesis, we decided to create one k-d tree per neuron. What we believed to be an improvement because we will examine fewer points per neuron has turned out to be a flag because for some trees the query point is so far that we need to examine all the trees. A solution to this problem could be using a lower amount of k-d trees such as 1 to 5 and changing the query algorithm from the nearest neighbour to a range search. We did some preliminary checks, not shown in this thesis, implementing these improvements. As a result, the query time improved to logarithmic in the number of neurons instead of lineal. Also, the time was reduced by one order of magnitude. But the heuristics did not show any significant improvement.

Chapter 6

Conclusion

6.1 Future Work

The next step in this research is to optimise the query algorithm to discard some branches of the k-d tree if the length in that dimension is too far from the query point. Also, future research should experiment with the number of k-d trees, because it will vary in performance. Finally, there should be more research on heuristics since it has helped in other fields and usually improves the computational cost. In other fields, researchers are introducing distributed computing due to its scalability compared to the price. So a distributed k-d tree must be taken into account for bigger and more realistic cases.

6.2 Conclusion

This study has investigated the performance of four heuristics for building a k-d tree, surface area heuristic, curve complexity heuristic, median of the hyperplane with maximum variance and minimum variance union. The results show that none of the heuristics increase the performance for the touchpoint task for one k-d tree per neuron. Thus, more tests are needed on different values of neurons per k-d tree, larger populations of neurons and mixed types of neurons, in order to find the most suitable one for whole-brain simulations.

Bibliography

[1] Adamsson, Marcus and Vorkapic, Aleksandar. "A comparison study of Kd-tree, Vp-tree and Octree for storing neuronal morphology data with respect to performance". PhD thesis. 2016. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-187026.

[2] Akar, Nora Abi, Cumming, Ben, Karakasis, Vasileios, Küsters, Anne, Klijn, Wouter, Peyser, Alexander, and Yates, Stuart. "Arbor — A Morphologically-Detailed Neural Network Simulation Library for Contemporary High-Performance Computing Architectures". In: (Feb. 2019), pp. 274–282. ISSN: 2377-5750. DOI: 10.1109/EMPDP.2019.8671560.

[3] Ascoli, Giorgio A. "Mobilizing the base of neuroscience data: the case of neuronal morphologies". en. In: *Nat. Rev. Neurosci.* 7.4 (Apr. 2006), pp. 318–324.

[4] Ascoli, Giorgio A, Donohue, Duncan E, and Halavi, Maryam. "NeuroMorpho.Org: a central resource for neuronal morphologies". en. In: *J. Neurosci.* 27.35 (Aug. 2007), pp. 9247–9251.

[5] Bentley, Jon Louis. "Multidimensional Binary Search Trees Used for Associative Searching". In: *Commun. ACM* 18.9 (Sept. 1975), pp. 509–517. ISSN: 0001-0782. DOI: 10.1145/361002.361007. URL: https://doi.org/10.1145/361002.361007.

[6] Blum, Manuel, Floyd, Robert W., Pratt, Vaughan, Rivest, Ronald L., and Tarjan, Robert E. "Time bounds for selection". In: *Journal of Computer and System Sciences* 7.4 (1973), pp. 448–461. ISSN: 0022-0000. DOI: https://doi.org/10.1016/S0022-0000(73)80033-9. URL: https://www.sciencedirect.com/science/article/pii/S0022000073800339.

[7] Brask, Anton and Berendt, Filip. "Analyzing the scalability of R*-tree regarding the neuron touch detection task". PhD thesis. 2020. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-279973.

[8] Carnevale, Nicholas T and Hines, Michael L. *The NEURON book.* Cambridge University Press, 2006.

[9] Chire. *Visulization of a 3D R-tree using ELKI.* File: `RTree-Visualization-3d.svg`. 2010. URL: `https://commons.wikimedia.org/wiki/File:RTree-Visualization-3D.svg`.

[10] Finkel, R A and Bentley, J L. "Quad trees a data structure for retrieval on composite keys". en. In: *Acta Inform.* 4.1 (1974), pp. 1–9.

[11] Friedman, Jerome H, Bentley, Jon Louis, and Finkel, Raphael Ari. "An algorithm for finding best matches in logarithmic expected time". en. In: *ACM Trans. Math. Softw.* 3.3 (Sept. 1977), pp. 209–226. ISSN: 0098-3500. DOI: `10.1145/355744.355745`. URL: `https://doi.org/10.1145/355744.355745`.

[12] Fuchs, Henry, Kedem, Zvi M., and Naylor, Bruce F. "On Visible Surface Generation by a Priori Tree Structures". In: SIGGRAPH '80 (1980), pp. 124–133. DOI: `10.1145/800250.807481`. URL: `https://doi.org/10.1145/800250.807481`.

[13] Guttman, Antonin. "R-Trees: A Dynamic Index Structure for Spatial Searching". In: *SIGMOD Rec.* 14.2 (June 1984), pp. 47–57. ISSN: 0163-5808. DOI: `10.1145/971697.602266`. URL: `https://doi.org/10.1145/971697.602266`.

[14] Herculano-Houzel, Suzana. "The remarkable, yet not extraordinary, human brain as a scaled-up primate brain and its associated cost". In: *Proceedings of the National Academy of Sciences* 109.supplement_1 (2012), pp. 10661–10668. DOI: `10.1073/pnas.1201895109`. eprint: `https://www.pnas.org/doi/pdf/10.1073/pnas.1201895109`. URL: `https://www.pnas.org/doi/abs/10.1073/pnas.1201895109`.

[15] Hill, Sean L., Wang, Yun, Riachi, Imad, Schürmann, Felix, and Markram, Henry. "Statistical connectivity provides a sufficient foundation for specific functional connectivity in neocortical neural microcircuits". In: *Proceedings of the National Academy of Sciences* 109.42 (2012), E2885–E2894. DOI: `10.1073/pnas.1202128109`. eprint: `https://www.pnas.org/doi/pdf/10.1073/pnas.1202128109`. URL: `https://www.pnas.org/doi/abs/10.1073/pnas.1202128109`.

[16] Hu, Linjia, Nooshabadi, Saeid, and Ahmadi, Majid. "Massively parallel KD-tree construction and nearest neighbor search algorithms". In: (May 2015), pp. 2752–2755. ISSN: 2158-1525. DOI: `10.1109/ISCAS.2015.7169256`.

[17] Hu, Linjia, Nooshabadi, Saeid, and Ahmadi, Majid. “Massively parallel KD-tree construction and nearest neighbor search algorithms”. In: *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2015, pp. 2752–2755. DOI: 10.1109/ISCAS.2015.7169256.

[18] ISO. *ISO/IEC 14882:2011 Information technology — Programming languages — C++*. Geneva, Switzerland: International Organization for Standardization, Feb. 2012, 1338 (est.) URL: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50372.

[19] Keller, Daniel, Erö, Csaba, and Markram, Henry. “Cell densities in the mouse brain: A systematic review”. en. In: *Front. Neuroanat.* 12 (Oct. 2018), p. 83.

[20] KiwiSunset. *A visualization of results obtained by running the Python kd-tree-construction program on [(2,3), (5,4), (9,6), (4,7), (8,1), (7,2)].* File: Kdtree$_2$*d.svg*. 2006. URL: https://commons.wikimedia.org/wiki/File:Kdtree_2d.svg.

[21] Knuth, Donald E. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Third. Boston: Addison-Wesley, 1997. ISBN: 0201896842 9780201896848.

[22] Kozloski, James and Wagner, John. “An Ultrascalable Solution to Large-scale Neural Tissue Simulation”. In: *Frontiers in Neuroinformatics* 5 (2011). ISSN: 1662-5196. DOI: 10.3389/fninf.2011.00015. URL: https://www.frontiersin.org/article/10.3389/fninf.2011.00015.

[23] Lu, Yucheng, Cheng, Luyu, Isenberg, Tobias, Fu, Chi-Wing, Chen, Guoning, Liu, Hui, Deussen, Oliver, and Wang, Yunhai. “Curve Complexity Heuristic KD-trees for Neighborhood-based Exploration of 3D Curves”. In: *Computer Graphics Forum* 40.2 (2021), pp. 461–474. DOI: https://doi.org/10.1111/cgf.142647. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.142647.

[24] MacDonald, J David and Booth, Kellogg S. “Heuristics for ray tracing using space subdivision”. en. In: *Vis. Comput.* 6.3 (May 1990), pp. 153–166.

[25] Markram, Henry, Meier, Karlheinz, Lippert, Thomas, Grillner, Sten, Frackowiak, Richard, Dehaene, Stanislas, Knoll, Alois, Sompolinsky, Haim, Verstreken, Kris, DeFelipe, Javier, Grant, Seth, Changeux, Jean-Pierre, and Saria, Alois. “Introducing the Human Brain Project”. In: *Procedia Computer Science* 7 (2011). Proceedings of the 2nd European Future Technologies Conference and Exhibition 2011 (FET 11), pp. 39–42. ISSN: 1877-0509. DOI: https://doi.org/10.1016/

j.procs.2011.12.015. URL: https://www.sciencedirect.com/science/article/pii/S1877050911006806.

[26] Meagher, Donald. *Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer.* Oct. 1980.

[27] Mulders, W H, West, M J, and Slomianka, L. "Neuron numbers in the presubiculum, parasubiculum, and entorhinal area of the rat". en. In: *J. Comp. Neurol.* 385.1 (Aug. 1997), pp. 83–94.

[28] Nü. *Schematic drawing of an octree, a data structure of computer science.* File: Octree2.png. 2006. URL: https://commons.wikimedia.org/wiki/File:Octree2.png.

[29] Physiopedia. *Interneurons — Physiopedia.* [Online; accessed 10-May-2022]. 2022. URL: https://www.physio-pedia.com/index.php?title=Interneurons&oldid=304286.

[30] Reimann, Michael W, King, James G, Muller, Eilif B, Ramaswamy, Srikanth, and Markram, Henry. "An algorithm to predict the connectome of neural microcircuits". en. In: *Front. Comput. Neurosci.* 9 (Oct. 2015), p. 120.

[31] Reimann, Michael W., King, James G., Muller, Eilif B., Ramaswamy, Srikanth, and Markram, Henry. "An algorithm to predict the connectome of neural microcircuits". In: *Frontiers in Computational Neuroscience* 9 (2015). ISSN: 1662-5188. DOI: 10.3389/fncom.2015.00120. URL: https://www.frontiersin.org/article/10.3389/fncom.2015.00120.

[32] Sah, Nirnath and Sikdar, Sujit K. "Transition in subicular burst firing neurons from epileptiform activity to suppressed state by feedforward inhibition". In: *European Journal of Neuroscience* 38.4 (2013), pp. 2542–2556. DOI: https://doi.org/10.1111/ejn.12262. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/ejn.12262. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/ejn.12262.

[33] Sherwood, Chet C, Miller, Sarah B, Karl, Molly, Stimpson, Cheryl D, Phillips, Kimberley A, Jacobs, Bob, Hof, Patrick R, Raghanti, Mary Ann, and Smaers, Jeroen B. "Invariant Synapse Density and Neuronal Connectivity Scaling in Primate Neocortical Evolution". In: *Cerebral Cortex* 30.10 (June 2020), pp. 5604–5615. ISSN: 1047-3211. DOI: 10.1093/cercor/bhaa149. eprint: https://academic.

oup.com/cercor/article-pdf/30/10/5604/38848884/bhaa149.pdf. URL: https://doi.org/10.1093/cercor/bhaa149.

[34] Stimberg, Marcel, Brette, Romain, and Goodman, Dan FM. “Brian 2, an intuitive and efficient neural simulator”. In: *eLife* 8 (Aug. 2019). Ed. by Frances K Skinner, e47314. ISSN: 2050-084X. DOI: 10.7554/eLife.47314.

[35] Tao, Yufei, Papadias, Dimitris, and Sun, Jimeng. “The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries”. In: VLDB ’03 (2003), pp. 790–801.

[36] Trujillo-Estrada, Laura, Dávila, José Carlos, Sánchez-Mejias, Elisabeth, Sánchez-Varo, Raquel, Gomez-Arboledas, Angela, Vizuete, Marisa, Vitorica, Javier, and Gutiérrez, Antonia. “Early neuronal loss and axonal/presynaptic damage is associated with accelerated amyloid-β accumulation in AβPP/PS1 Alzheimer’s disease mice subiculum”. en. In: *J. Alzheimers. Dis.* 42.2 (2014), pp. 521–541.

[37] Wald, Ingo and Havran, Vlastimil. “On building fast kd-Trees for Ray Tracing, and on doing that in O(N log N)”. In: (Sept. 2006), pp. 61–69. DOI: 10.1109/RT.2006.280216.

Appendix - Contents

Appendix A

Source Code

All the source code and documentation needed to repeat the results from this project can be found on my personal repository on github.