# THE RELATION BETWEEN THE COMPUTATIONAL EFFORT AND PATH LENGTH OF A* PATHFINDING IN AN OCTREE REPRESENTATION OF AN INDOOR POINT CLOUD.

A research proposal submitted to the Delft University of Technology in partial fulfilment of the requirements for the degree of Master of Science in Geomatics

by

Olivier Rodenberg

February 2016

# 1 INTRODUCTION

Path finding is computing an optimal route between a starting point and a goal. One of the most important and challenging aspects is finding a collision free path. Path finding is well established in 2D outdoor situation, think of car navigations or the navigations applications on modern smart phones [Noto and Sato, 2000]. Recently there is more and more interest for indoor path planning. Although this has been researched in the field of robotics in the 80 and 90 these algorithms are mainly focussed on 2D navigation. Besides the algorithms were developed for relatively slow robots. Nowadays we would like to have the ability to help people navigate inside buildings. Or send a drone inside a building when this is to dangerous for people. Or help boats navigate through waterways. What all these examples have in common is that an object with a certain geometry needs to find an optimal collision free path between a starting point and goal.

To manage this we, at least, need to know the geometry of the object and a model of the environment. One way of representing the latter one can be with a point cloud. A point cloud is a large collection of point having at least an X, Y and Z coordinate representing a boundary of space. Figure 1.1 shows a point cloud of the Bouwpub.

**Figure 1.1:** Point cloud of the Bouwpub.

The model of the environment alone does not give enough information to find a route, for this the empty (pointless) space is needed. To derive the empty space from a point cloud some changes to the representation need to be made. One common method for doing this is by creating an octree of the point cloud. Creating an octree from a point cloud involves placing the point cloud in minimal bounding box. This box is recursively subdivided “into eight congruent disjoint cubes (called octants)” this continuous until each cube is either full, empty or until a pre defined level is reached[Samet, 1988]. This way of partitioning a point cloud results in a hierarchical subdivision of space. This make operations like neighbour finding and indexing really

efficient [Vörös, 2000]. Figure 1.2 shows how the subdivision an quadtree, the 2D counterpart of an octree.

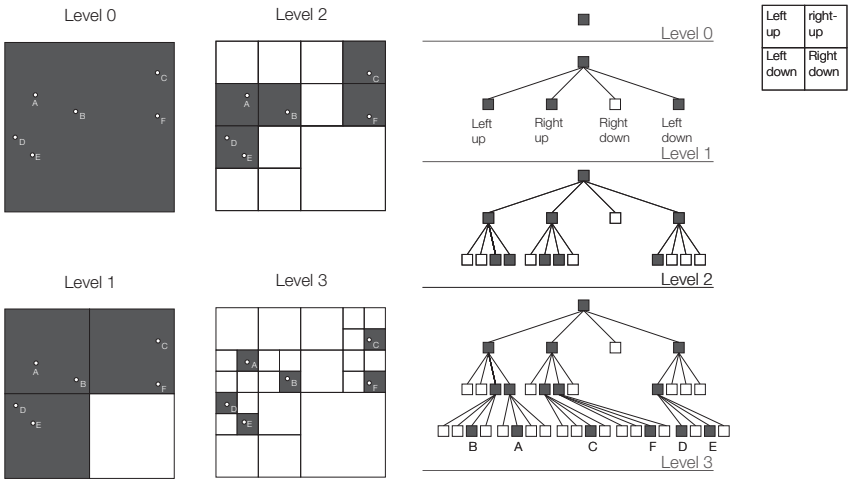


**Figure 1.2:** an visualization of a quadtree

In an octree a large empty space can be represented by a large node high in the octree(see figure 1.3). These large empty nodes reduce the amount of nodes in the octree. This is actually the main advantage of using an octree in pathfinding. Because there are less nodes and thus less neighbours to inspect. This reduces the amount of routes to discover and subsequently the computations in a pathfinding operation.

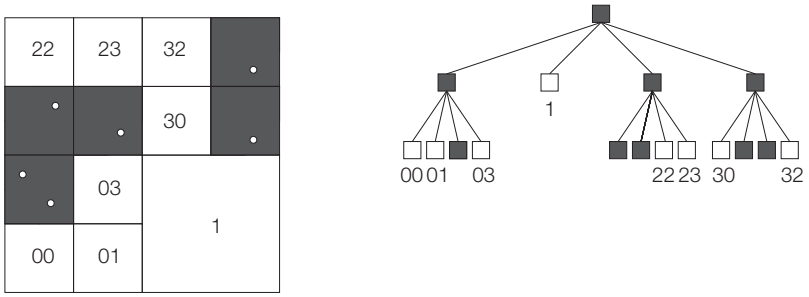


**Figure 1.3:** *For visualization purposes a quadtree is used instead of an octree to explain the concept.* Large empty spaces can be represented by a node which is high in the quadtree. The left image shows the a quadtree with some black nodes and white nodes. The right image shows the location of these nodes in the octree. Notice that a large area in the left image refers to a higher location in the octree.

Broersen et al. [2015], in a project called pointless, created a work flow to identify the empty space inside a point cloud using an octree data structure. And a rudimentary pathfinding algorithm using this octree data structure was developed (see section 2.2.4).

In many use cases pathfinding algorithm can be useful. For instance, a fire department wants to assess a situation in a building but does not want to send in a fire fighters. For this situation a drone could be used. Before a drone can move inside the situation it should know what path to

follow. A second case study could be the routing of autonomous robots through a warehouse. These robots need to travel the same route many times, therefore it is key that this route is a short as possible. To manage this the calculated route should be of a certain resolution. In emergency situation an indoor pathfinding method can be used to direct persons to the closest emergency exit. In an hospital it can be used to direct patients to the room of their doctors appointment. Further, an indoor pathfinding application can be used for automatic monitoring of building. A unmanned airborne vehicle can be used to inspect area hard to reach areas in an automated way, think of inspecting the condition of pipes on the celling. Even in underwater situations it is very useful to have an pathfinding method generating a collision free route [Guang-lei and He-Ming, 2012]. In situations where a cable need to be route between a two points a pathfinding application can be used [Fragoso Rodrigues et al., 2014].

## 1.1 SCIENTIFIC RELEVANCE

Often assumptions are made in a design processes. For example, having less nodes in an octree reduces the amount of computation in a pathfinding application. However, the exact influence of an assumption is often ignored. and therefore not known. This is also the case in pathfinding applications. This thesis will research the effect of assumptions made in A* pathfinding in an octree representation of an indoor point cloud. It will identify the exact impact of components effecting A* pathfinding on the computational complexity and path length. So far no research has been conducted identifying the effects of these components in A* pathfinding in an octree data representation of an indoor point cloud.

# 2 RELATED WORK

This section presents an overview of related work for the different aspects relevant for this research.

## 2.1 OCTREE GENERATION

The procedure described bellow was developed by Broersen et al. [2015].

The octree implemented by Broersen et al. [2015] is a linear octree, which means it is stored in a linear array instead in a tree. To refer to the nodes in the octree each one has a location code. Important to note is that the octree is generated from an indoor point cloud. The method created by Broersen et al. [2015] can roughly be divided into three steps. First the point cloud is pre processed. Secondly the non-empty (black) nodes are computed. And finally the empty (white) nodes are derived from the non-empty nodes [Broersen et al., 2015].

The first step involves processing the point cloud. The point cloud is translated so its origin lays in the coordinate (0,0,0). The point cloud is scaled so each node in the octree has a coordinates between integer 0 and integer 2n [Broersen et al., 2015].

Next the non-empty nodes are computed. The non-empty nodes are generated by searching the location of each point in the octree. Because all nodes in the octree have a coordinate between integer 0 and 2n the points can be snapped to a node by removing its decimals. Next the location code of each node needs to be computed. Because an octree is generated by recursive subdivision of cubes into eight children nodes we can encode each level with a single number between 0 and 7 (see figure 2.1). The location code is a string of numbers indicating the path through the octree to reach a certain node, figure 2.2 shows the process of numbering nodes in an quadtree. To have the highest possible resolution of the empty space all points are forced to split to the maximum octree depth. To obtain the location code a technique called binary masking is used. Each point starts with an empty location code. Binary masking basically checks per split level on which side of the split a node lays, this is done for the X, Y and Z direction. If this is known the octant where the node lays is located, see figure 2.3. The number of this octant is concatenated to the location code. This is repeated until the maximum split level is reached. The end product is a string of numbers for each point referring to the location code. In the end a list with the location code of all non-empty nodes is created [Broersen et al., 2015].

The final step is to derive the empty nodes from the non-empty nodes. To find this a list containing all possible location codes of the octree is compared

with the list of non-empty location codes. The nodes which are not in the list are the empty nodes. Both the location codes of the non-empty and empty nodes are stored in a database [Broersen et al., 2015].

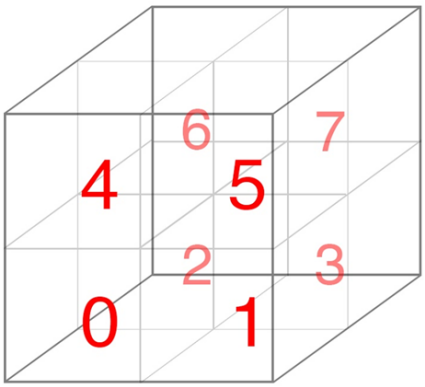


**Figure 2.1:** Numbering of the octree, source: [Broersen et al., 2015]

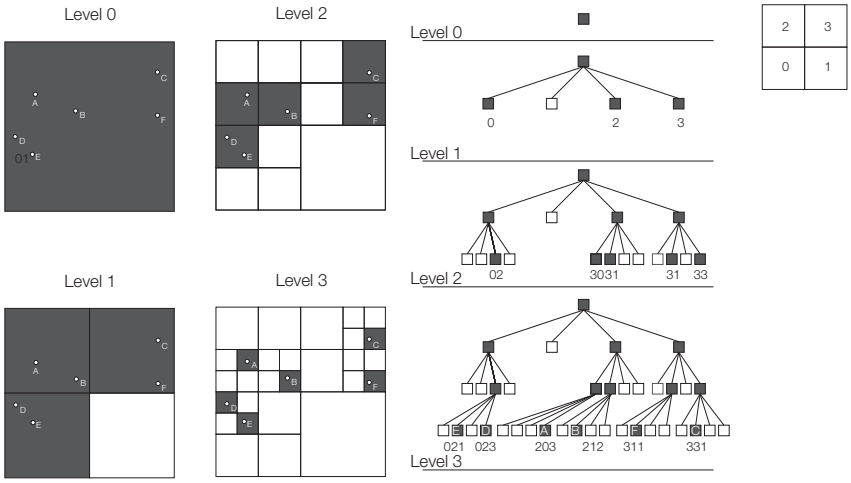


**Figure 2.2:** Numbering of the non-empty nodes in a quadtree

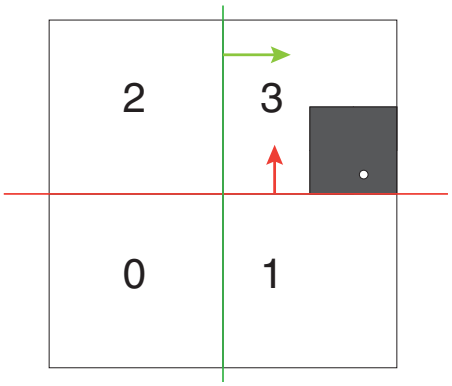


**Figure 2.3:** *(For visualization purposes a quadtree is used instead of an octree to explain the concept.)*The non-empty(black) node lays right of the green split line and above of the red split line, thus is lays in quadrant 3.

## 2.2 FEATURES OF PATHFINDING ALGORITHMS

In this section different kind of pathfinding algorithms are explained.

### 2.2.1 Hill-climbing

Hill-climbing is one of the easiest methods, but it does have the tendency to get stuck in U shaped obstacles. There are multiple Hill-climbing methods, two common methods are random descent and steepest descent. The basic idea of steepest descent is to select the neighbour, to become the next node, of a current node which is closest to the target. And repeat until the target is reached. In random descent a random neighbour is selected to turn into the next node (see figure 2.4).Because the method constantly searches for the neighbour with the smallest cost it can get stuck inside a U shaped obstacle [Dowsland, 1991].

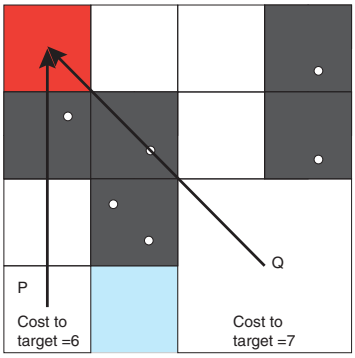


**Figure 2.4:** The cost from node $P$ to the target is smaller than that of $Q$ therefore $p$ is select as next node from the current node(blue).

Herman [1986] presents a method where the pathfinding start with hill climbing, if this gets stuck in an obstacle an A* algorithm (this algorithm is explained in section 2.2.4) takes over until hill-climbing can continue. This process is repeated until the target node is reached. This method has a time advantage, however, as hill climbing can move away from an optimal path. In Vörös [2001] a steepest descent method is used in combination with an distance map. The distance map is calculated in an octree. By using an octree the amount of memory needed is greatly reduced.

### 2.2.2 Tracing

In general the Tracing pathfinding algorithm moves in the direction of the target until it reaches a obstacle. The best way to describe what happens is to images walking aside this obstacle. While walking aside the obstacle the algorithm decides when to part sides with this obstacle. One method for doing this is to stop tracing when the entity is moving in the direction of the target. The main problem with this method is that it takes more time tracing than necessary [Stout, 1996] [Xu, 2012]. This method is not likely to work in an octree generated from an point cloud due to the many obstacles in a indoor point cloud.

### 2.2.3 Dijkstra

A common pathfinding algorithm is described by Edsger Dijkstra in 1959 [Nosrati et al., 2012]. The Dijkstra algorithm calculates the optimal route between a start and the target node by searching for the minimal travel cost. The algorithm works as follows: It appends all neighbouring nodes of the starting node to an open set and calculates the cost to get there. The node with the lowest accumulated cost is sent to a closed set. The neighbours of this node are searched for and placed in the open set. And these steps are repeated until the target node is placed in the closest list. When a node already exists in the closed list it will only replace the existing one if the accumulated cost is lower, else it will be ignored.

In general this process will be concentric. Due to this the time to find the shortest is heavily dependent on the network size and distance between the start node and the target node. With a large distance the search time becomes very long. This makes the Dijkstra algorithm unsuitable for real-time applications [Noto and Sato, 2000] and thus not usable for the octree structure.

### 2.2.4 A*

A solution for the time problem of the Dijkstra algorithm was found by Hart et al. [1968]. They extended Dijkstra's solution by implementing a heuristic cost, an approximation, to the algorithm. One way of using the heuristic cost is to estimates the cost from the current node to the target node. The total cost ($f(c)$) for each neighbour($c$) is now:

$$f(c) = g(c) + h(c) \tag{2.1}$$

Where $g(c)$ is the cost from the start node to the node $c$. And $h(c)$ is the heuristic cost from $c$ to the target [Nosrati et al., 2012] [Kambhampati and Davis, 1985].

Because the heuristic is added to $f(c)$ all nodes which are not between the start node and the goal node have a higher heuristic cost compared to the nodes which are not. Therefore, the speed of the algorithm is not related to the network size, but only to the length of the route. Figure 2.5 shows the difference between the algorithm of Dijkstra and A*.

Xu et al. [2015] describes a pathfinding method inside an octree data structure using an A* algorithm. They find the neighbours of the current node by checking for each node in the octree if the length of the does not exceed half the length of the sum of the lengths of the two octants. If so, they are adjacent. Obviously this is an expensive calculation, so this is probably not the best method of finding neighbours. They state that by using an octree structure the amount of nodes to be checked is greatly reduced, this makes it a lot faster than voxel based approach.

In Broersen et al. [2015] a pathfinding algorithm was developed which is based on an A* algorithm. In this algorithm a route is searched through the empty space of the octree. The method searches for the route with a minimal distance. Two nodes are neighbours if they share a common face. Because in an octree the node are not all of the same size a method based on Vörös [2000] was used to search for the neighbours. In this method smaller, larger and of equal size for searched. The neighbours are computed on the fly. The algorithm did not take into account the object size.

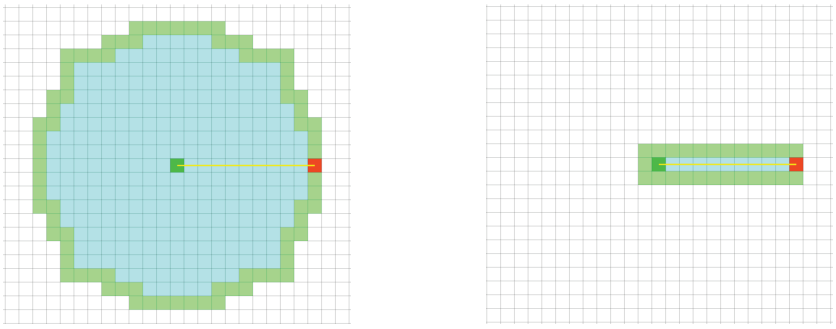

**Figure 2.5:** The left image shows a a route computed between the start(green node) and a goal (red node) using the Dijkstra algorithm. Al blue nodes have been checked in the algorithm. The right image shows the same but for an A* algorithm. Due to the Heuristic cost in the A* algorithm the amount of nodes processed is much smaller compared to the Dijkstra algorithm. source: Xu [2012]

### 2.2.5 Potential field

In a potential field approach an artificial field is generated in which both the target node and all obstacles direct a force on each white node in a field. The target node has a attractive force and the obstacles have an repulsive force. These forces are strong at the source and gradually decrease as the distance to the source increase (see figure 2.6). The sum of these forces are the potential value of a node, together all nodes create a potential field. The maximal potential field is in the obstacles, a low potential field is thus favourable in a pathfinding applications. To actually find a path in the potential field an other pathfinding algorithm is needed [Hou and Zheng, 1994] [Hwang and Ahuja, 1992].

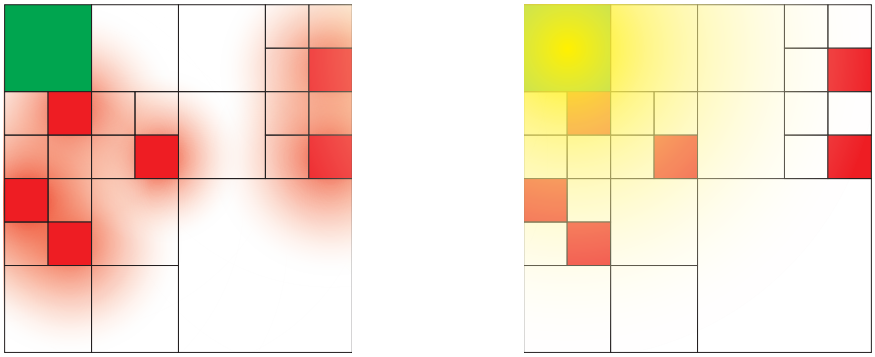

**Figure 2.6:** *(For visualization purposes a quadtree is used instead of an octree to explain the concept.)* Nodes close to the target get a high attractive force and are more desirable

### 2.2.6 Multi resolution

In Kambhampati and Davis [1985] a multi resolution finding method in a quadtree is proposed. They propose to navigate in a single level of the

octree. This is done to minimize the amount of neighbours which have to be searched. Because there are less nodes in a higher level than all leaf nodes (see figure 2.7. For this to work grey nodes need to be accepted as a neighbour. A grey node is a non-leaf node with at least one empty and one non-empty descendants(see image 2.8). To know if it is possible to navigate through a grey node three questions are formulated.

1. ”what is done when one of the neighbours of the current node is grey?
2. how is the current expanded when the current best node is a grey node?
3. how is the grey node processed to get to the final that contains free nodes exclusively?” [Kambhampati and Davis, 1985]

Further, they present the requirements for each question to asses whether a grey node can be used in the. Figure 2.9 shows the difference between a route computed at a pruned level and one in the leaf nodes.

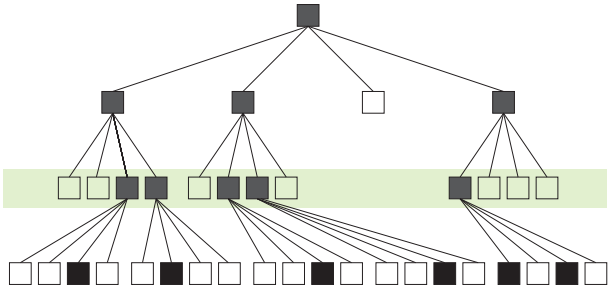

**Figure 2.7:** Only the nodes in a single level, the green zone, are used as possible neighbours. This method allows for grey nodes to be valid neighbours.

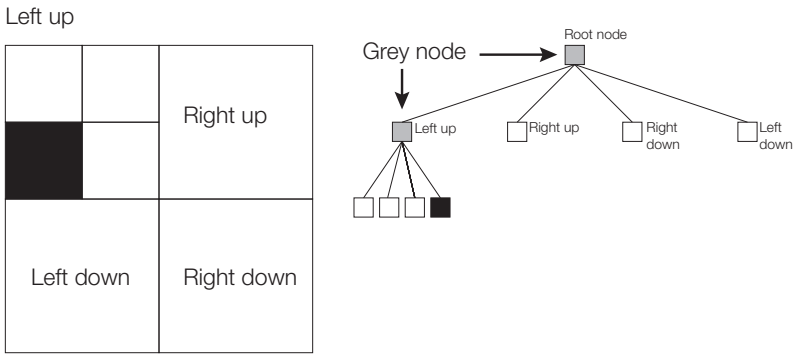


**Figure 2.8:** *(For visualization purposes a quadtree is used instead of an octree to explain the concept.)*This figure shows two grey nodes. The ’left up’ node is grey because it has one black descendant and three white descendants. The root node is grey because the ’left up’ node is grey and its other descendants are white.

Besides the multi resolution planning a method to avoid collisions is presented. In a preprocessing stage, adjacent black nodes (obstacles) are grouped. Next, a buffer is created around the grouped black nodes. This buffer has is at least half the size of the entity which has to navigate in the octree. So is

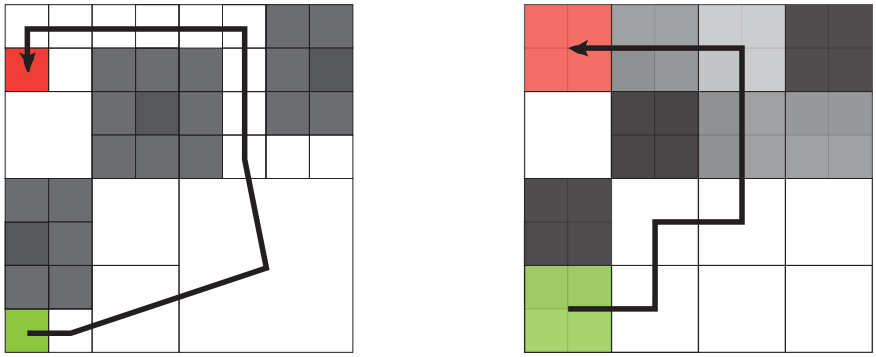

**Figure 2.9:** *(For visualization purposes a quadtree is used instead of an octree to explain the concept.)*The left image shows the through the total quadtree, and the right image through a single level in the quadtree.

it ensured that the object can not collide with any obstacles. Finally, a method a distance transform is computed, this calculates the minimal distance from the centre of a node to the closest boundary of an black node (obstacle). This distance map is later used to indicate the clearance of each white node.

## 2.3 NEIGHBOUR FINDING

To find a route in an octree the path connectivity between the nodes in an octree need to be known.The type of path connectivity between nodes in the octree defines when two nodes are neighbours. There are three types of connections, neighbours having a common: face, edge or vertex. Combining these connection types leads to different types of connectivities, figure 2.10 shows the types of connectivities.

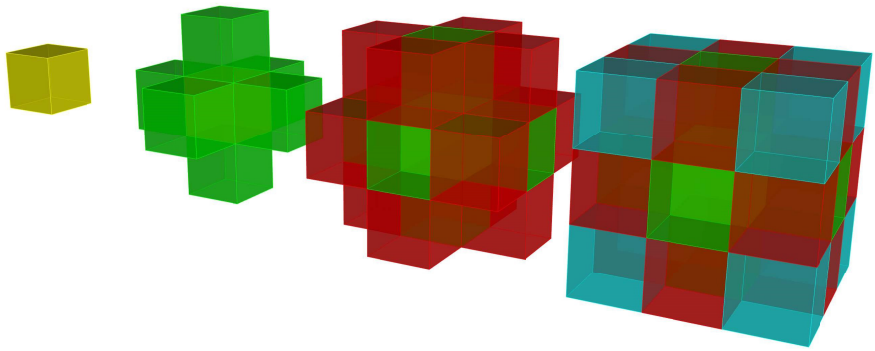

**Figure 2.10:** From left to right: The first image shows an node, the next image shows neighbours with a common face (6 connectivity), the next image shows neighbours with a common face and edge (18 connectivity), the last image shows neighbours having a common face, edge and vertex (26 connectivity)

There are different solutions to find neighbours nodes in an octree. One method described by Samet [1989a] suggest to search a for common ancestors by ascending the tree, if this is found the tree is descended in search of

the neighbouring nodes. The advantage of this method is that all possible neighbours are computed. This means neighbour sharing a common: face, edge and vertex (see figure 2.10). However, this method does not take into consideration, that nodes outside the octree do not exist. Vörös [2000] solves this problem by stating: the root level is the highest common ancestor. However, this method only computes neighbours sharing a common face.

# 3 RESEARCH OBJECTIVES

This chapter will describe the research objectives. In section 3.1 the research question and sub questions are presented. Next, the motivation is given, here a purpose statement for the questions is provided. And finally the scope of the research is presented.

## 3.1 RESEARCH QUESTION

The main research question for this proposal is:

*What is the relation between the computational effort and path length of A* pathfinding in an octree representation of an indoor point cloud?*

The main research question has three components which can influence the computational effort and path length, these are: processing operations on the point cloud which are important for the generation of the octree, the octree properties and the components in the A* pathfinding algorithm. Therefore the following sub questions are formulated:

- *What point cloud processing operations are important for the generation of the octree and what is their effect?*
- *What octree properties influence the computational effort and path length and what is their effect?*
- *What components in the A* algorithm influence the computational effort and path length and what is their effect?*

## 3.2 MOTIVATION

Often assumptions are made in a design processes. For example, an octree with a maximum depth of six levels instead of eight has less nodes. Therefore it is assumed to find a route faster(in computing time) compared to one with a maximum depth of eight levels. However, the exact influence of this assumption is not known. This research aims to identify components effecting the computational effort and path length. Furthermore, it will analyse the impact of these components. Knowing these effects will give insight in how to design a pathfinding method in an octree with a specific purpose.

## 3.3 RESEARCH SCOPE

The main goal of the thesis is to formulate recommendations on how to set up the different components for a specific pathfinding .

In this research one point cloud dataset will be used. The differences between point cloud generation techniques will not be researched. The differences between different point cloud acquisition devices will not be researched.

Although the world is not static, in this research the model of the world is assumed to be static.

Section 2.1 describes the way an octree is constructed from a point cloud in this research. Except for processing operations on the point cloud, which are researched in the second sub question, this method will remain the same.

In the research an A* pathfinding algorithm will be used. Differences between other pathfinding methods will not be researched.

The geometry of the object which needs to navigate through the empty space of the octree will be generalized. The geometry will be generalized to an octant. Furthermore, the objects are able to translate but not rotate.

# 4 METHODOLOGY

This chapter will describe the methodology used in the research. First the general method is explained, in here the procedure for all the test is presented. Next the methodology of the three sub questions is provided.

## 4.1 GENERAL

### 4.1.1 Collision avoidance

All objects which are navigated in real world situations have a certain object size. For a pathfinding method to be usable in a real world situation it needs find a collision free path considering the object size. A path is collision free if the object does not intersect with all obstacles (black node). A collision occurs when the centre point of a node in the path is less than half the object size away from an obstacle.

One way of preventing collision is to create a buffer around each black cell. This buffer needs to be at least half the size of the object (see figure 4.1). To be certain collisions are not possible a buffer of all 26 neighbours need to be created (see figure 2.10. To find all 26 equal sized neighbours of a node the method of Samet [1989b] can be used. In his approach the neighbours can be inner (having the same ancestor node) or outer (having a different ancestor node). The method searches for common ancestors by ascending the tree, if this ancestor is found the tree is descended in search of the neighbouring nodes. The disadvantage of this method is that it only works for objects having the same size. If a larger object needs to navigate through the empty space of the octree the buffer no longer suffices. As a consequence the octree needs to be recomputed with a buffer half the size of the object.

An approach where one dataset is usable for all object sizes is described by Samet [1982a]. He describes a distance transform which calculates for each node the minimal distance to a boundary between a black and white cell in a quadtree. The method is based on a theorem which states: " for any black block in the image , its neighbours cannot all be black. Proof: If all of the neighbours are black merging will take place and the block would not exist " (see image 4.2). Because it is not possible that all 8 neighbours are of the same kind the border between a white and black node has to be between one of the eight neighbours. This means only the neighbours have to be checked to find the closest border (see figure 4.3).

Performing this operation for all nodes results in an distance map where each node has a value indicating the minimal distance to a border. This value can be used as a constraint in an A* pathfinding algorithm ensuring an object can only pass a node if it is large enough. Besides, this constraint can be used to navigate with a minimal distance to obstacles.

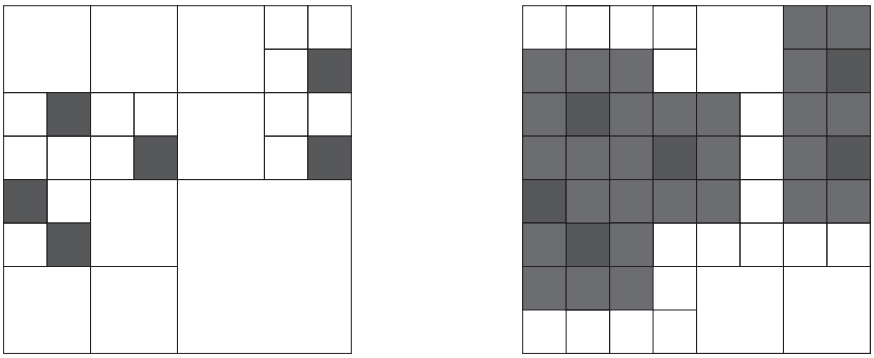

**Figure 4.1:** *(For visualization purposes a quadtree is used instead of an octree to explain the concept.)* The left image shows the situation without a buffer, in the right image a buffer is created around each black node. This buffer must be at least half the object size. this way an UAV can navigate through the white nodes without any collisions.

The latter method will be implemented due to its extended use for all object sizes. And because it can be used as constraint so an object is not allowed come to close to an obstacle.

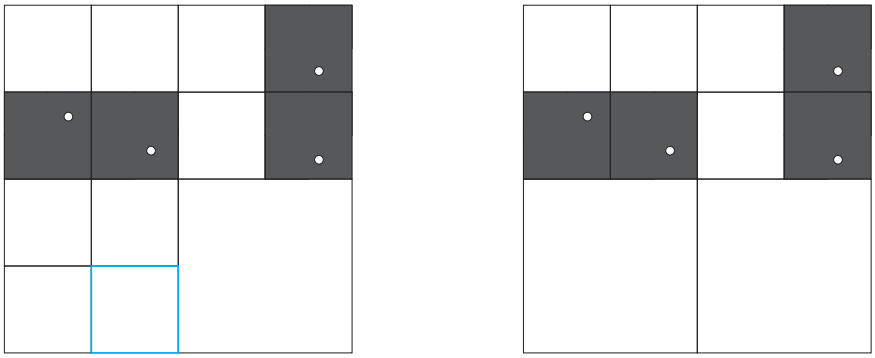

**Figure 4.2:** If all neighbours of the octant highlighted in blue, in the left image, are of the same colour 4 neighbouring octants having the same ancestor will merge (see right image).

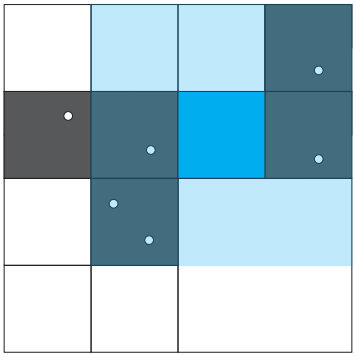

**Figure 4.3:** The border between blue and the closest black node must lie in the blue area.

## 4.2 NEIGHBOUR FINDING

Important in an A* algorithm is the neighbour finding method. The method in which neighbour will be searched will be based on [Vörös, 2000] and [Samet, 1982b]. These methods search for neighbours: bigger, smaller and of equal size. In the methods the neighbours can be of inner (having the same ancestor node) or outer (having a different ancestor node). The method searches for common ancestors by ascending the tree, next the tree is descended in search of the neighbouring nodes.

By default nodes are neighbours if they share a common face (see figure 2.10).

### 4.2.1 Set up and storage

To get an idea of the effect on the computational effort and path length benchmark test will be to be performed. Each component is going to be tested separately, this means all other component are constant during the test. Doing this ensures that other component can not influence the test results.

Each component needs to be tested on a range of start and goal sets. The locations of these start and goal sets will be selected in the point cloud dataset. With the exact location the location code for the nodes can be computed.

To analyse the test all results must be stored. All results will be stored in a PostgreSQL database.

## 4.3 POINT CLOUD PROCESSING OPERATIONS

The goal of this step is to find the effects of point cloud processing operations on the generation of the octree and pathfinding. The operations which are going to be researched are: translation and rotation. These operations may effect the time to compute the octree and the amount of nodes in the octree. Besides, the effect an octree generated with or without a translated point cloud is researched on the computation effort and path length in A* pathfinding. Both the effects on octree generation and pathfinding are going to be stored in the database.

First the effect of translation on the generation of an octree is researched. For this two versions of the point cloud dataset are created, a translated point cloud and the original (not translated) point cloud. The translated version of the point cloud is created by translating the origin of the bounding box of the point cloud to coordinate (0,0,0) (see figure 4.4). An octree is generated of both these point cloud datasets. The time to computation and amount of octants are stored in the database.

Next te difference between the two octrees on A* pathfinding is researched. For this a path needs to be searched in both octrees. Important is that the location of the start and goal points are identical related to the point cloud in both octrees. To ensure this the location of the start and goal point are selected in the original point cloud. The get the location of these points in the translated point cloud the coordinates are translated the same as the point cloud. The locational code of these points is then computed using the

method described in 2.1. With these location code the A* pathfinding can search for a path. Of this path the: computation time, amount of iterations, path length and actual path are stored in the database.

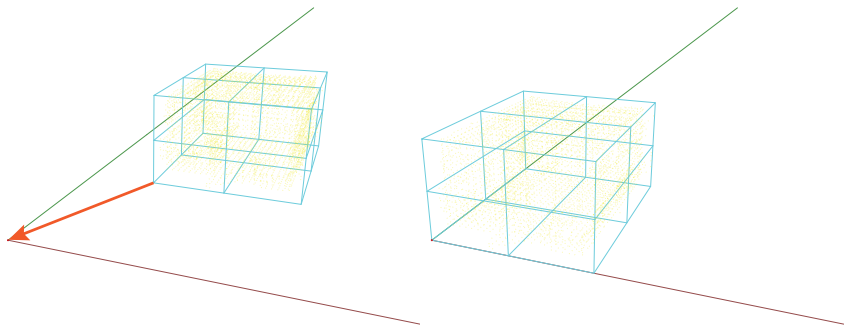

**Figure 4.4:** The origin of the bounding box(blue) of the point cloud(yellow) is translated to coordinate (0,0,0) of the field.

The method described above is almost the same for rotation, the only difference is that instead of translating the point cloud it is rotated. The original point cloud is rotated so the floor follows the $x$ and $y$ plane. And the walls follow the $x$ and $y$ axes as much as possible (see figure 4.5). The rotation will be performed in CloudCompare.

For researching the effects of pathfinding the method is the same. The start and goal points of the rotated dataset are rotated the same way as the point cloud to ensure identical locations in the octree related to the point cloud.

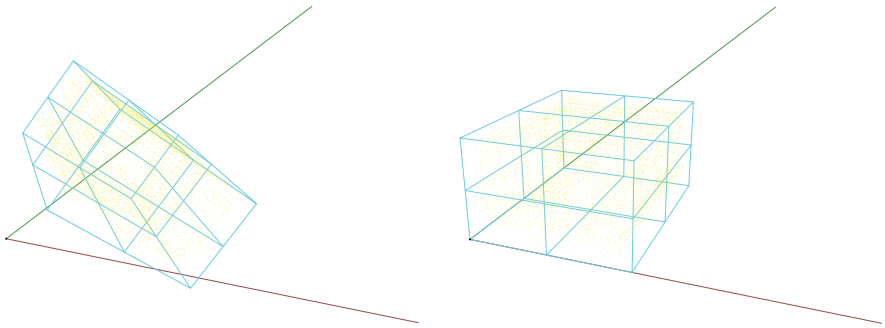

**Figure 4.5:** The left image shows a point cloud(yellow) which is not alligned with the axis. This point cloud is rotated so it follows the $x$ and $y$ plane and the axis(see right image)

## 4.4 OCTREE PROPERTIES

The goal of this step is to research the effect octree properties have on the computational complexity and path length in A* pathfinding. The two properties which are researched are: The octree depth and the path connectivity between the nodes in an octree.

### 4.4.1 Tree depth of the octree

The depth of an octree refers to the maximum split level of the octree. This depth effects both the computational effort as the route length of a pathfinding method. The goal of this step is to research what these effects are. For each maximum depth an octree needs to be computed. For each octree the computation time and amount of octants are stored in the database.

Next the effect on the path is researched. For this a path between a start and goal point is computed for each octree. Because the depths of the octrees differ so does the subdivision of space. This means the location code of the start and goal node may be different in each octree. So the locational code of the start and goal point need to be computed for each octree with a different tree depth. The method to get a locational code from a point is described in section 2.1. This method ensures that the path in each octree with a different depth has the same start and goal location related to the point cloud. For each path the: computation time, amount of iterations, path length and actual path are stored in the database.

### 4.4.2 Type of path connectivity between nodes in the octree

The goal of this step is to identify the exact effect the different type of connectivities have on the computational effort and path length. The three types of connectivity described in figure 2.10 will be researched. A pathfinding command will be given for each type of connectivity. Important is that in each command the start and end node are identical. For each path the: computation time, amount of iterations, path length and actual path are stored in the database.

## 4.5 A* COMPONENTS

The goal of this step is to identify the effect between types of heuristic costs in A* pathfinding.

Like described in section 2.2.4 the A* algorithm introduces an heuristic cost to the total cost. There are more than one ways to calculate the heuristic distance [Xu, 2012]. First the different types of heuristic distance ways need to be identified. Next the differences between these types need to be tested. This is done by running an A* pathfinding command with identical start and goal nodes for each of the heuristic distance methods. For each path the: computation time, amount of iterations, path length and actual path are stored in the database.

# 5 SCHEDULE

Table 5.1 shows the important dates provided by the academic graduation calender.

**Table 5.1:** Academic graduation calender

| Activity | Date |
|---|---|
| **P1 - draft project plan + presentation** | 09-11-2015 |
| **P2 - final project plan + presentation** | 14-01-2016 |
| **P3 - mid-term process presentation** | 04-04-2016 to 15-04-2016 |
| **P4 - draft thesis + presentation** | 09-05-2016 to 20-05-2016 |
| **P5 - final thesis + public presentation** | 13-06-2016 to 24-06-2016 |

## 5.1 GANTT

Figure 5.1 the schedule to meet the research objectives.

## 5.2 MEETINGS

Meetings will be held with my first supervisor Prof.dr. Sisi Zlatanova and second supervisor ir. Edward Verbree to provide guidance in the process.

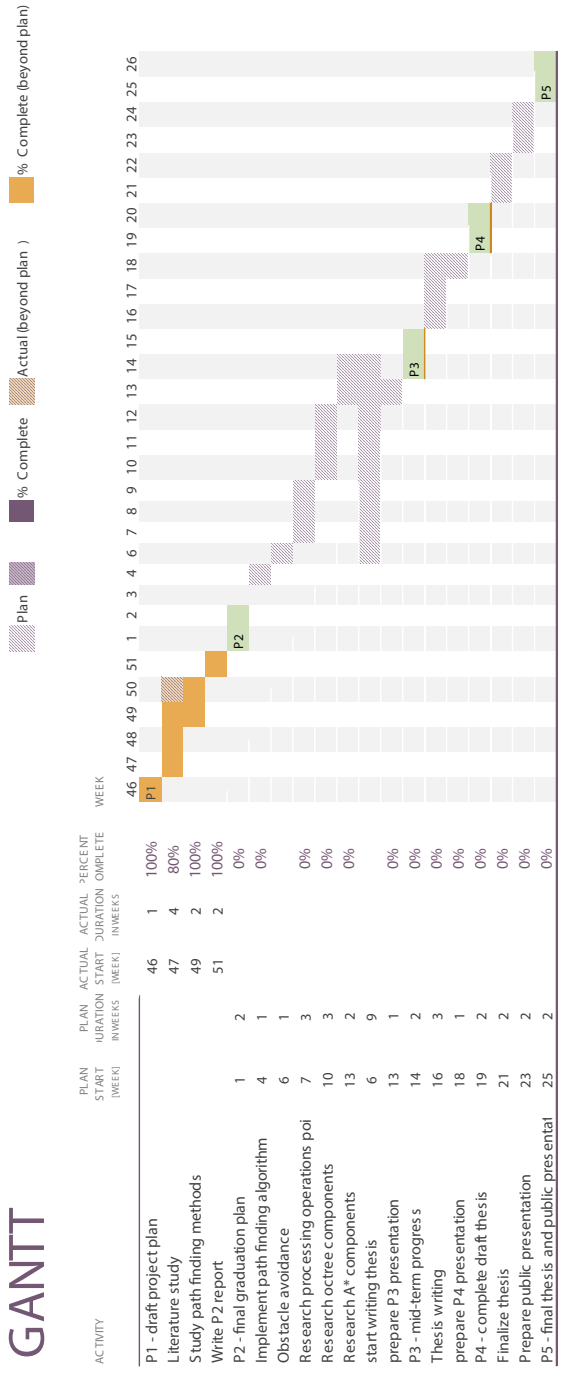


**Figure 5.1:** The GANTT chart for this graduation thesis.

# 6 TOOLS AND DATA

## 6.1 DATA

There are three datasets are multiple point cloud datasets available. Two point clouds of a single room: one of the 'slaapzaal' of the museum 'het Prinsenhof' and one of the 'Bouwpub" at the faculty of Architecture in Delft. There are two point cloud of the fire department in Berkel en Rodenrijs. One is made by the mobile Zeb1 laser scanner and covers the entire building. The other is made by the stationary Leica C10 and covers only the top floor. And finally there is a mesh generated by the google Tango of the same fire department. A point cloud can be created from this mesh by exporting only the nodes of the mesh. The dataset which is going to be used in the initial stage of the research is the point cloud generated by the zeb1. This is the point cloud that covers an entire building.

## 6.2 TOOLS

For writing the reports *latex* is going to be used. To clean the point clouds *CloudCompare* is going to be used. For the extraction of the point cloud form the mesh generated by the google Tango *meshlab* is going to be used. For scripting the *python* is used. The Database management system *PostgreSQL* in combination with *PostGIS* are used. To connect Python to PostgreSQL the python extension *psycopg2* is used. For the visualization of the routes *Potree* and/or *3ds Max* are going to be used. For potential animations *3ds Max* is going to be used.

# BIBLIOGRAPHY


Broersen, T., Fichtner, F., Heeres, E., de Liefde, I., and Rodenberg, O. (2015). Project pointless. identifying, visualising and pathfinding through empty space in interior point clouds using an octree approach. Geomatics Synthesis Project.

Dowsland, K. A. (1991). Hill-climbing, simulated annealing and the steiner problem in graphs. *Engineering Optimization*, 17(1-2):91–107.

Fragoso Rodrigues, S., Bauer, P., Bosman, P., and Pierik, J. (2014). Collection system cable routing and wake losses optimization in offshore wind farms. In *XIII SEPOPE: Symposium of Specialists in Electrical Operational and Expansion Planning, Foz do Iguassu, Brasil, 18-21 May 2014*.

Gargantini, I. (1982). Linear octrees for fast processing of three-dimensional objects. *Computer Graphics and Image Processing*, 20(4):365–374.

Guang-lei, Z. and He-Ming, J. (2012). Global path planning of auv based on improved ant colony optimization algorithm. In *Automation and Logistics (ICAL), 2012 IEEE International Conference on*, pages 606–610. IEEE.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107.

Herman, M. (1986). Fast, three-dimensional, collision-free motion planning. In *Robotics and Automation. Proceedings. 1986 IEEE International Conference on*, volume 3, pages 1056–1063. IEEE.

Hou, E. S. and Zheng, D. (1994). Mobile robot path planning based on hierarchical hexagonal decomposition and artificial potential fields. *Journal of Robotic Systems*, 11(7):605–614.

Hwang, Y. K. and Ahuja, N. (1992). A potential field approach to path planning. *Robotics and Automation, IEEE Transactions on*, 8(1):23–32.

Kambhampati, S. and Davis, L. S. (1985). Multiresolution path planning for mobile robots. Technical report, DTIC Document.

Kim, J. and Lee, S. (2009). Fast neighbor cells finding method for multiple octree representation. In *2009 IEEE International Symposium on Computational Intelligence in Robotics and Automation - (CIRA)*.

Namdari, M. H., Hejazi, S. R., and Palhang, M. (2015). MCPN, octree neighbor finding during tree model construction using parental neighboring rule. *3D Research*, 6(3).

Nosrati, M., Karimi, R., and Hasanvand, H. A. (2012). Investigation of the*(star) search algorithms: Characteristics, methods and approaches. *World Applied Programming*, 2(4):251–256.

Noto, M. and Sato, H. (2000). A method for the shortest path search by extended dijkstra algorithm. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 3, pages 2316–2320. IEEE.

Payeur, P. (2006). A computational technique for free space localization in 3-D multiresolution probabilistic environment models. *IEEE Trans. Instrum. Meas.*, 55(5):1734–1746.

Samet, H. (1982a). Distance transform for images represented by quadtrees. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (3):298–303.

Samet, H. (1982b). Neighbor finding techniques for images represented by quadtrees. *Computer Graphics and Image Processing*, 18(1):37–57.

Samet, H. (1988). An overview of quadtrees, octrees, and related hierarchical data structures. In *Theoretical Foundations of Computer Graphics and CAD*, pages 51–68. Springer.

Samet, H. (1989a). Neighbor finding in images represented by octrees. *Computer Vision, Graphics, and Image Processing*, 45(3):400.

Samet, H. (1989b). Neighbor finding in images represented by octrees. *Computer Vision, Graphics, and Image Processing*, 46(3):367–386.

Stout, B. (1996). Smart moves: Intelligent pathfinding. *Game developer magazine*, 10:28–35.

Vörös, J. (2000). A strategy for repetitive neighbor finding in octree representations. *Image Vis. Comput.*, 18(14):1085–1091.

Vörös, J. (2000). A strategy for repetitive neighbor finding in octree representations. *Image and Vision Computing*, 18(14):1085–1091.

Vörös, J. (2001). Low-cost implementation of distance maps for path planning using matrix quadtrees and octrees. *Robotics and Computer-Integrated Manufacturing*, 17(6):447–459.

Wang, M. and Tseng, Y. (2011). Incremental segmentation of lidar point clouds with an octree-structured voxel space. *The Photogrammetric Record*, 26(133):32–57.

Xu, S., Honegger, D., Pollefeys, M., and Heng, L. (2015). Real-time 3d navigation for autonomous vision-guided mavs. IROS.

Xu, X. (2012). Pathfinding.js.

Zhou, K., Gong, M., Huang, X., and Guo, B. (2011). Data-Parallel octrees for surface reconstruction. *IEEE Trans. Vis. Comput. Graph.*, 17(5):669–681.