École Polytechnique Fédérale de Lausanne

From Play-Doh to Roombots

Radosław Dryzner

Sciper : 234218
radoslaw.dryzner@epfl.ch

supervised by
Mehmet Mutlu
Dr. Stéphane Bonardi
Simon Hauser

Biorobotics Laboratory
Prof. Auke Jan Ijspeert

January 8, 2016

Acknowledgements

I would like to thank the faculty of Computer Sciences of EPFL as well as Prof. Ijspeert and the entire Biorobotics Laboratory for giving me the opportunity of doing and completing such a great and interesting project.

I would also like to thank my supervisors Mehmet Multu, and also Dr. Stéphane Bonardi and Simon Hauser, for helping me throughout the project and making it possible to complete it. It wouldn't have been possible without their advices and knowledge.

I would like to thank the professors of which course material gave me the necessary skills and knowledge to leading this project into completion.

Finally, I would like to thank my parents for helping me and supporting me through this semester and project.

Summary

The Play-Doh to Roombots project is a software development project. The software in question is used to create a Roombots structures out of a Play-Doh sculptures made in real world. This consists of scanning a Play-Doh sculpture into the software and using different approaches to reconstruct it using Roombots.

During the development of the software, multiple approaches were used with varying performance of time and quality of reconstruction. These approaches consist of artificial intelligence search methods such as Breadth-First-Search, Depth-First-Search and Greedy algorithms on one hand, as well as a direct voxelization method on the other. These modular approaches leave the possibility for extension in the future.

During the project, difficulties of the given problem have been identified and multiple solutions have been developed.

Contents

Chapter 1

Introduction

The goal of the Play-Doh to Roombots project is to provide an interactive and intuitive way of generating Roombots structures.

Roombots are modular self-reconfigurable robots developed at the Biorob lab here at EPFL. Robots of this kind can change their morphology for different needs. The final goal of the Roombots project is to make modular structures autonomously, like furniture in a room for example. Roombots would be able to change the furniture of a room on the fly on request of a user. The goal of this semester project is to give users an easy way of telling these robots what to construct.

Roombots structures are complex constructions. They can be viewed as essentially a voxel cloud with the property of being constructable by Roombots modules, since a single Roombots module consists of two cube-like parts. A user has to give this voxel cloud so that we know where a Roombots should be placed. This can be a non-trivial task, especially to someone not familiar with computer systems. It is akin to playing with blocks and making things out of it, yet in the unfamiliar world of 3D modelling on a computer. In this project, we give an easy implementation that anyone could use to create Roombots structures.

Play-Doh is a toy that children use from the youngest ages. It consists of dough-like material which is easy to shape and rigid enough to keep the given shape when no external force is applied to it. It is a very easy way of building something intuitively. It has the advantage of being something you can visualize and modify in real world. In this project, we will see how we can get Roombots structures out of such Play-Doh sculptures.

The question we are asking ourselves here is whether it is possible to obtain a Roombots structure in reasonably small amount of time and resources, out of a sculpture made in Play-Doh. This project has three main aspects:

1. 3D scanning of a real object into a virtual environment
2. Voxelization of the 3D object in the virtual environment
3. Filling the 3D model with Roombots modules

The presence of 3D scanning technology today in the form of video game consoles, such as Kinect, as well as software using it, makes it very easy to convert the real world Play-Doh structure into the computer for further processing. For this part of the project, we will use an existing software, while the other two aspects has been developed especially for the project.

Voxelization of 3D meshes is another process that is well known to be accessible. Our project resembles voxelization, while assuring certain properties to make sure the voxel cloud is constructable using Roombots. Voxelization is useful for us because it helps us quantify our final results in terms of number of voxels covered by the structure.

To fill the 3D model with Roombots modules, we have developed four different approaches that we will look into in detail. For this project, we are ignoring the alignment of the modules. We assume all degree of freedom angles are 0°. In other words, the Roombots are on a discrete grid.

Through these three aspects, it will be possible for us to build Roombots structures in reasonable time and resource use out of real world objects such as Play-Doh sculptures.

While the project focuses on Play-Doh, it will be also entirely possible to use any kind of real world object (as long as they can be 3D scanned), or any kind of 3D mesh, and reconstruct them into a Roombots structure, as a result of this project.

Literature Review

In the field of providing intuitive and easy way of manipulating Roombots, there is one approach that have been studied here at EPFL[1]. The approach gave the users a way of showing the Roombots where to go on a grid using pointing gestures. In this project however, we are focusing on the shape of the final Roombots structure, not the position of individual modules. Instead of showing where the Roombots should go, they are told what they should be making.

When it comes to recreating an original mesh with voxel-like structures, another case was done for the use of Lego bricks[2]. In that case, the construction was done per layer because the bricks themselves can only be placed flat on a layer. In our case, Roombots can be vertical as well, but the research done on Lego-bricks served as inspiration for this project's work.

Chapter 2

Methods

As we have seen before, this project has three main aspects: 3D scanning of real object into a virtual environment, voxelization of the 3D object in the virtual environment and filling the 3D model with RoomBots modules. We will see these steps in detail now as well as the different approaches of filling the 3D model which are either artificial intelligence search methods such as Breadth-First-Search, Depth-First-Search and Greedy algorithms, or a direct voxelization method with verification.

Figure 2.1 represents every step of this process as a flowchart.

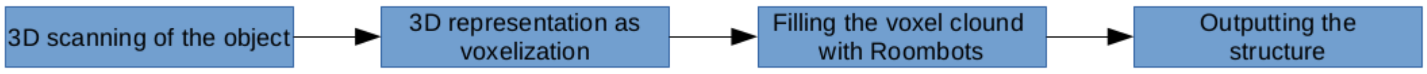
3D scanning of the object
3D representation as voxelization
Filling the voxel clound with Roombots
Outputting the structure

Figure 2.1: Flowchart representing the aspects of the project

2.1 Scanning the Play-Doh sculpture

To do any kind of processing with the Play-Doh sculpture, we first need to import it into our software systems. For this end, a Kinect sensor has been used.

Kinect sensors are made for the video-game consoles XBox 360 and XBox One[3]. This sensor was chosen for various reasons. It is one of the oldest sensors that has been available at large for everyone to use, and because of that, there has been already a great deal of software and techniques that allow anyone to scan 3D objects as meshes using it. In this project, we use one of such software.

The software that we are using is KScan3D[4]. This software allows easy scanning of objects. The way we do this, is that we place the object on support that can be manually rotated while the Kinect remains stationary. The software captures the scene geometry when requested, and it can capture multiple times from different point of views, automatically aligning the captures. This allows us to manually rotate the support and capture the object from many angles.

Figure 2.2 shows the scanned object where 24 scans were used to capture it. You can see the presence of holes in the mesh there but also on figure 2.3 where the whole bottom part of the mesh is open.

Figure 2.4 shows the automatically finalized mesh as outputted by the software. The mesh still has some holes but many were already closed.

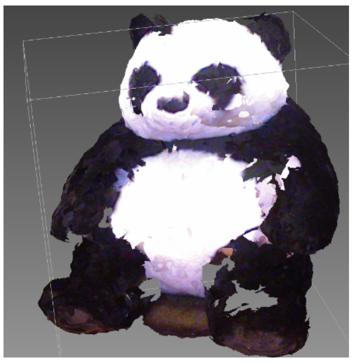

Figure 2.2: Combined scans of the object

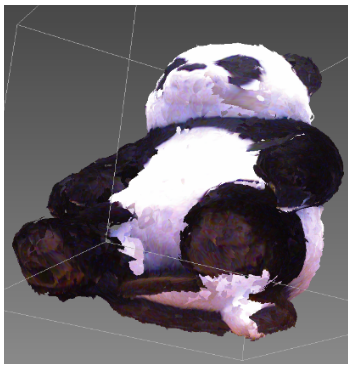

Figure 2.3: Combined scans of the object

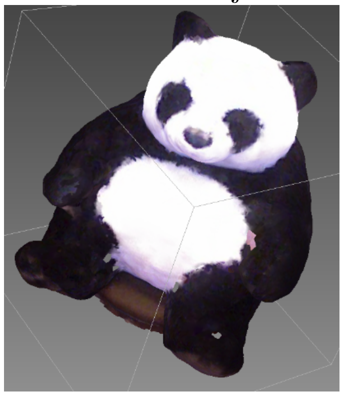

Figure 2.4: Finalized and combined mesh

The next step is to remove any unnecessary parts that have been scanned with the object. For this, a 3D modelling software is used, in this case, Meshmixer[5]. Using it, the additional parts, such as the support which could also be scanned, are removed from the final mesh.

Finally, we close the mesh if there are any holes present inside of it using the same software as shown in the figure 2.5. Finally, we decrease the number of vertices on the mesh so that we can remove noise of the 3D scanner. The final mesh is shown in the figure 2.6.

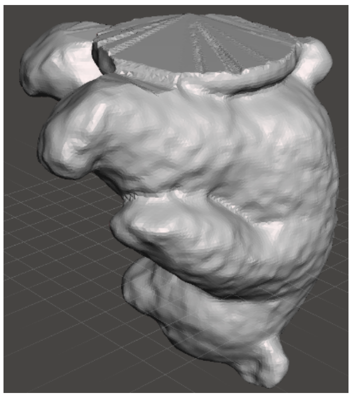
Figure 2.5: Solidified mesh

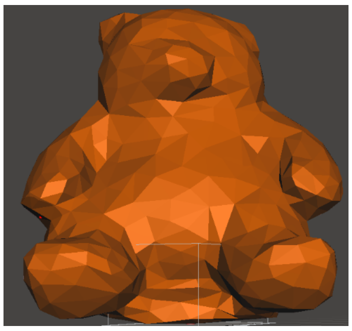
Figure 2.6: Finalized mesh

2.2 Voxelization of the scanned mesh

The scanned mesh is now voxelized. This is done for multiple reasons.

The voxelization is essential for determining the fidelity of the constructed structure to the original mesh. Having voxels instead of a cloud of vertices, it is possible to precisely quantify the number of voxels that are inside or outside our output structure.

During the course of the project, two different voxelization approaches has been used. At first, another software was used[6], but in the end, the voxelization has been integrated in the project software to better adapt to the needs of the project.

The first approach using the voxelizer software[6] worked in the following way. A 3D mesh could be loaded in the program. The parameter that we used there was the voxel resolution. This resolution represented the number of voxels on the longest axis of the object. The software also permits visualization of the mesh and the result as a voxel cloud. Figure 2.7

shows us the voxel cloud over the mesh and figure 2.8 shows the same voxel cloud but without the original mesh.

Figure 2.9 illustrates the effect of reducing the voxel resolution. The voxel cloud doesn't represent the original mesh as well as before.

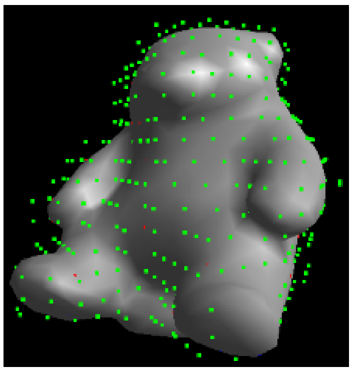
Figure 2.7: Original mesh with voxel cloud

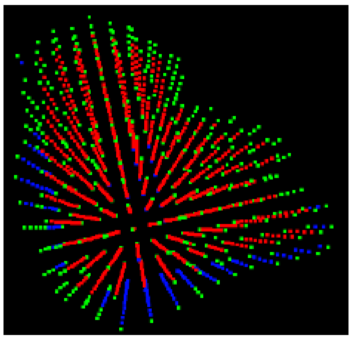
Figure 2.8: Voxel cloud without the original mesh

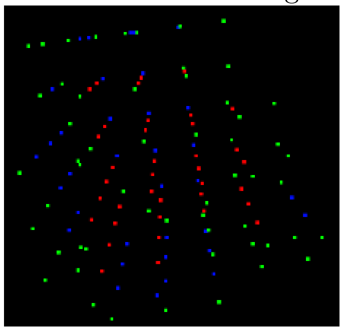
Figure 2.9: Voxel cloud with reduced resolution

The voxel resolution for figures 2.7 and 2.8 is 15 while for figure 2.9 it is 6.

The colours of the voxel points depend on their position relative to the original mesh. Green voxels are mostly outside the mesh, blue voxels are exactly on the mesh (their centre point is on the mesh surface), red voxels are entirely enclosed by the mesh.

For the second approach, the voxelization using our custom voxelizer is done as follows: the 3D mesh is placed on a discrete voxel grid, with voxel sized specified as a variable that can be modified. A loop then over all possible voxel locations determines whether a voxel should be placed there. For each voxel, this decision consists of a few steps.

1. A ray, or line, is cast from the potential voxel location into an arbitrary direction.
2. For each face of the 3D mesh, an intersection point with the line is calculated.
3. A check is made to see whether this point is actually in the bounds of the face in question, and if it is the case, there has been an intersection.
4. The count of intersections is checked. If it is odd, the voxel point is located inside the mesh, otherwise it is outside.
5. If the centre point of the voxel is inside, we place a voxel at this location. Voxels with their centre points outside of the mesh, but that are partially inside the mesh are not placed.

Figure 2.10 illustrates this process in 2D with the red voxel identified as outside the mesh because of 2 intersections with the mesh while the green one is inside because it has 1 intersection with the mesh.

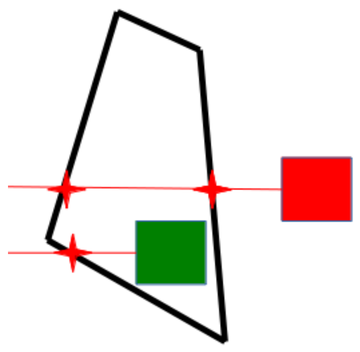

Figure 2.10: Identifying voxel candidates that are inside or outside of the mesh

The algorithm used to determine the intersection is the Möller-Trumbore intersection algorithm[7].

At the end of this step, we have a voxel cloud with which we can construct a Roombots structure. The different approaches used to achieve this are explained in the following subsections.

2.3 Filling the 3D model with RoomBots

This problem is similar to "Tiling Problems" but here we are looking for the "Best Tiling"

One important notion to define here is the size of a Roombots module. The size of the Roombots is defined as the size of one of the cubes it is made of. This parameter will affect the quality of the reconstruction in a similar manner as the voxelization resolution does for the voxelization of a 3D mesh. When this Roombot size decreases, it means we can fit more Roombots modules over the mesh, increasing the resolution of the reconstruction. This is why we will talk about increasing the resolution when decreasing the value of Roombots module's size. The size of a Roombots module can be given in two ways.

The first one is to directly specify the size of one cube making up the Roombots. This size is in the system of coordinates of the input file of voxels cloud/mesh, depending on the approach taken (whether one of the Artificial Intelligence search methods or Direct Voxelization with Verification).

The second approach is for the user to specify the desired size of the structure in real world values. The user can give the desired width, length or height of the final structure. Only one of these parameters can be given to keep the same proportions of the structure as in the input. This parameter is given in centimetres.

2.3.1 Artificial Intelligence Search Methods

The Artificial Intelligence search methods that we are using are Breadth-First-Search, Depth-First-Search and Greedy algorithms. These algorithms work on graphs representing tree structures. So we have to define what are the nodes of our tree, and how do we obtain them.

The search space here is the set of all possible Roombots structures. The search tree's nodes are Roombots structures. The structures at depth n of the search tree are made up of n Roombots.

The children nodes of a node represent the possible expansions of the structure at the parent node. In other words, if a structure is represented by node A, the children of A are all possible structures that can be created by connecting one Roombots module to the structure. Each children corresponds to a different position of this Roombots module. Figure 2.11 illustrates this in 2D showing the children nodes of a node and corresponding Roombots structures.

To obtain these children, we look at what new connections can be made with the already existing Roombots of the structure. For each Roombots, and for each connection point, we look at all possible Roombots we can put

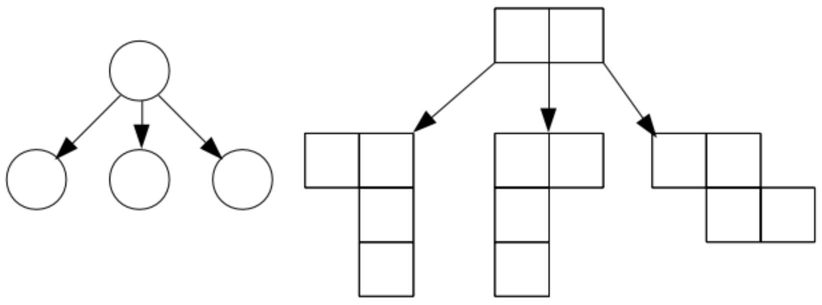

Figure 2.11: Children nodes and corresponding RoomBot structures. Note that not all children nodes are represented here.

at that point. We add it, opening a new search node, only if it satisfies a coverage criteria. That is a certain number of voxels covered. This is done to limit expansion of the search tree for new Roombots that would cover only one voxel for example. The lower limit used here is 55% of the maximum coverage possible. The maximum coverage possible is defined as the number of voxels a Roombots could cover if it was perfectly aligned with the voxel grid. For example, if one Roombots module can cover at most 20 voxels, we place a module if it covers at least 11 voxels in that candidate location.

To compare different structures and assign a quality metric for them, we define a value function for the nodes of our tree. This function takes a Roombots structure and the voxel cloud we are building the structure over, and then outputs the metric value of the structure. This metric is defined as:

$$value = \frac{\#voxelsCovered - \#voxelsNotCovered - \#proxyVoxels}{\#totalNumberOfVoxels}$$

The number of voxels covered and the number of voxels not covered are self explanatory. The proxy voxels are voxels that could be covered if they were there. In other words, if a Roombots structure only covers 55% of the maximum coverage possible, and another covers 100%, the second one should be valued more. So the proxy voxels for the first Roombots of the example is the remaining 45%. The ideal case for a structure is to have its metric equal to 1. It cannot be obtained however in most of the problems.

This function ensures that we penalize structures that cannot represent the voxel cloud well.

Since the Search Tree needs at least the starting node, we must first place the first Roombots module as our starting structure. The position of the first Roombots module to place has to be given manually by the user. This is done in the system of coordinates of the input voxel cloud/mesh.

Breadth-First-Search (BFS) construction

The first approach is the Breadth-First traversal of a search tree to construct the Roombots structure.

From the initially placed Roombots module the usual Breadth-First traversal of the tree is made. The algorithm is looking for leaf nodes for evaluation later on. The leaf nodes are the nodes that do not have any more children. In BFS, the algorithm walks through all children of every node at a given depth in the tree and collects all the leaf nodes. Once that depth is completed, the algorithm goes on the next depth and goes through all the nodes at that depth again.

As per definition of this traversal, the algorithm is exhaustive and considers every single Roombots structure possible over the voxel cloud. The best one of the leaf nodes is chosen after the algorithm finishes to obtain the best Roombots structure over the mesh. The best leaf is defined as the leaf having the highest value using our metric function.

Depth-First-Search (DFS) construction

Other than the Breadth-First-Search algorithm, there is also the Depth-First-Search algorithm. This algorithm works as expected, per the usual definition of Depth-First-Search. The algorithm chooses one child of a node and goes to it, then it does the same on that child, going deeper into the tree until it reaches a leaf, which it collects. Then it backs up to the parent of the leaf, taking another of its children, then repeats the process starting from that child, eventually backing up all the way to the starting node having traversed all nodes in the tree.

Since we are searching the entire search space, BFS and DFS are doing the same work in the end. The algorithm has been implemented for comparison and easy verification of either of the algorithm.

Greedy construction

This algorithm is the usual greedy algorithm. It works similarly to DFS, with two main differences:

1. The choice of the child to be traversed from a node is not arbitrary. In the greedy algorithm, we choose the child that has the highest value as defined by our metric function.

2. Once the algorithm reaches a leaf, it does not go up again, but outputs that leaf as its solution.

It is clear here that this algorithm does not visit every possible node, but just one branch of the search tree. This algorithm was implemented to see what quality of results it can obtain.

2.3.2 Direct Voxelization with Verification construction

The last approach that is used does not make use of any value function or any search algorithm because searching the entire search space takes too much time for big structures. The direct voxelization simply voxelizes the original mesh at the desired resolution, that is, the desired size of a Roombots module. Then it makes sure it is constructable by Roombots.

During the voxelization, newly placed voxels are paired with already present voxels to make up the Roombots. One of the the neighbours is chosen, prioritizing the x axis over the y axis over the z axis.
This is done so that at the end, the unpaired voxels can be removed from the structure. This is a step that we need to make because a voxel cloud is not necessarily constructable with Roombots which are made of two cubes.

Chapter 3

Results

We will now look into the results of each step. We will in particular look at the time resources of each approach for feasibility as well as visual quality of the reconstructed object, or its fidelity to the original mesh.

For this section, we will use the object shown on figure 3.1 to illustrate the results of each step, unless specified otherwise. More samples will be shown at the end of this chapter.

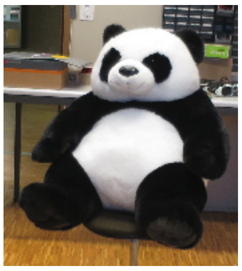

Figure 3.1: Real world object

All results were obtained on the same computer and in the same environment to ensure the lack of any result differences possible between machines of different performances.

3.1 Scanning the Play-Doh form

As explained in the Methods chapter, we can obtain scanned meshes such as the one on figure 3.2.

The vertex count of the mesh is 1200 and the face count is 1200. The scanning was done using 24 captures of the object as it was being rotated.

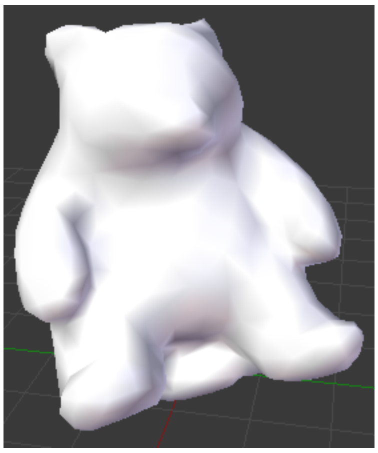

Figure 3.2: Scanned Mesh with corrections

The holes in the mesh were closed and vertex count decreased.

3.2 Voxelization of the mesh using our custom voxelizer

Taking the 3D mesh as the input of the voxelization, we obtain the result shown on figure 3.3

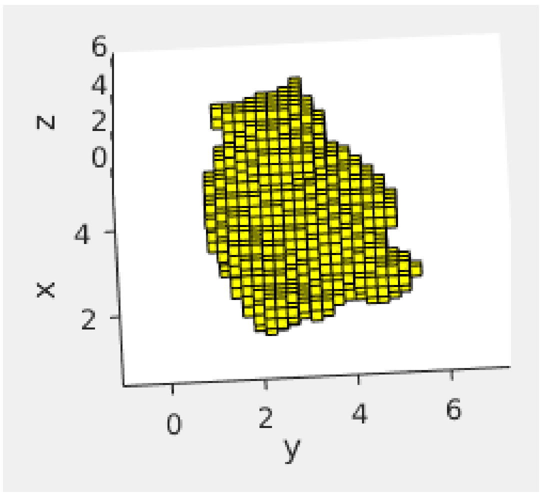

Figure 3.3: Voxelized mesh

The resolution of the voxel cloud can be specified as needed, here it has been chosen to be 0.24. The resulting voxel count is 2039, which gives

a rather faithful discrete reconstruction of the original mesh.
The construction of this voxel cloud took 1,6877 seconds on average of 10 samples.

The voxelization of the mesh scales with the resolution of the voxelization. The original mesh defines the bound of the discrete grid, while the resolution defines the granularity of the steps inside the grid.
In the figure 3.4 you can see a graph that shows the running time of the voxelization for the same mesh with varying resolution:

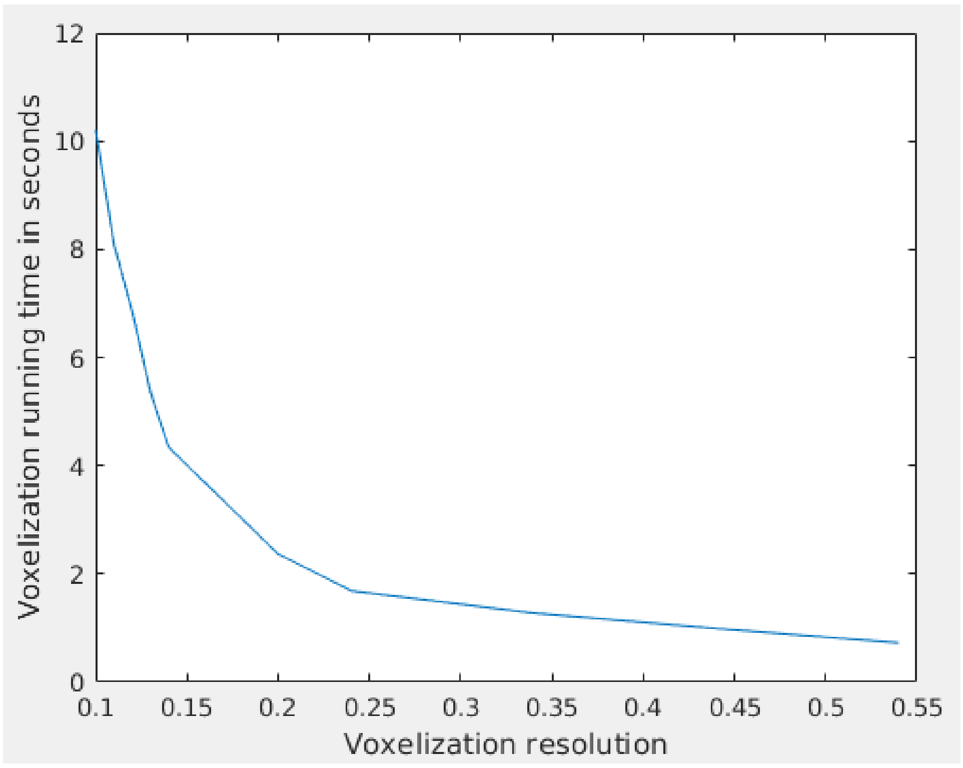

Figure 3.4: Voxelization run time over voxelization resolution

3.3 Breadth-First-Search and Depth-First-Search algorithms

Unfortunately, for these algorithms, we quickly run into problems where the program does not terminate fast enough to obtain meaningful results. On figure 3.5, you can see the result of the algorithm on the voxelized mesh from figure 3.3. The voxelization resolution was 0.24, and the Roombots size here was 1.2.

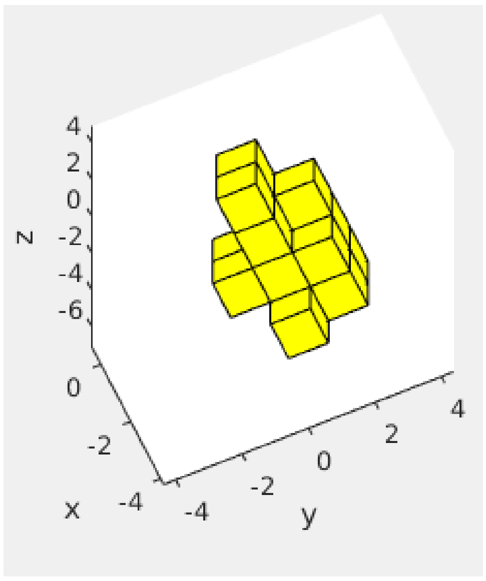

Figure 3.5: Result of Breadth-First-Search construction algorithm

This structure is made of 16 cubes or 8 Roombots. It was obtained using the Breadth-First-Search algorithm and it took 45.615 seconds to obtain it. The results from the Depth-First-Search algorithm are identical. The metric value of this structure is 0.45.

While this result is the best leaf of the search, we can also take a look at a middle quality one and the worst one in figures 3.6 and 3.7 respectively.

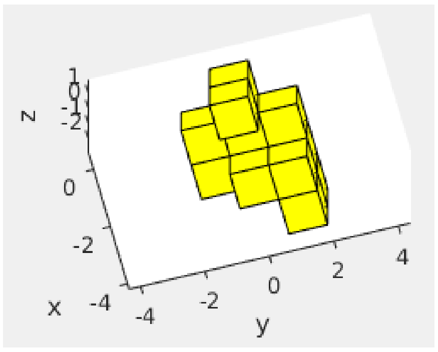

Figure 3.6: BFS result with metric value of 0.33 and 6 Roombots

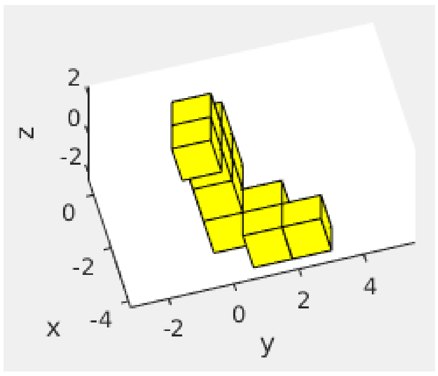

Figure 3.7: BFS result with metric value of 0.15 and 5 Roombots

To better visualize the evolution of the growth of the tree, we will look at the number of children opened in the tree at a given depth, here 4 is taken as the depth for figure 3.8.

Finally, to take a look at the execution time of each iteration of the algorithm as it traverses the tree, we can look at figure 3.9

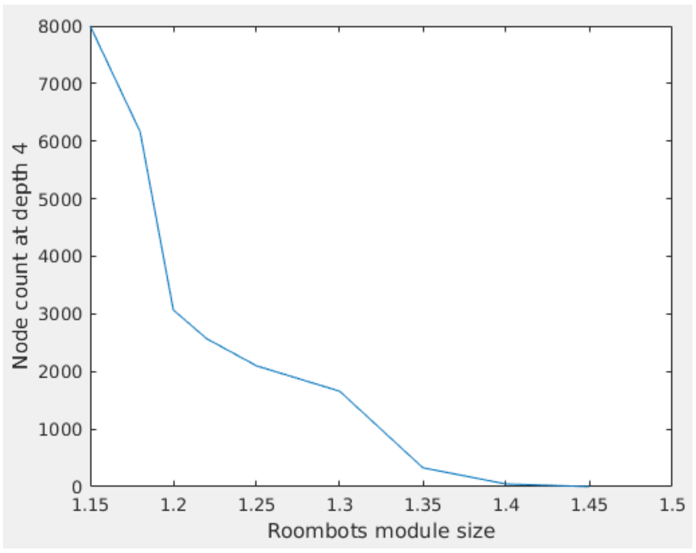

Figure 3.8: Child count created at depth 5 over the Roombots size

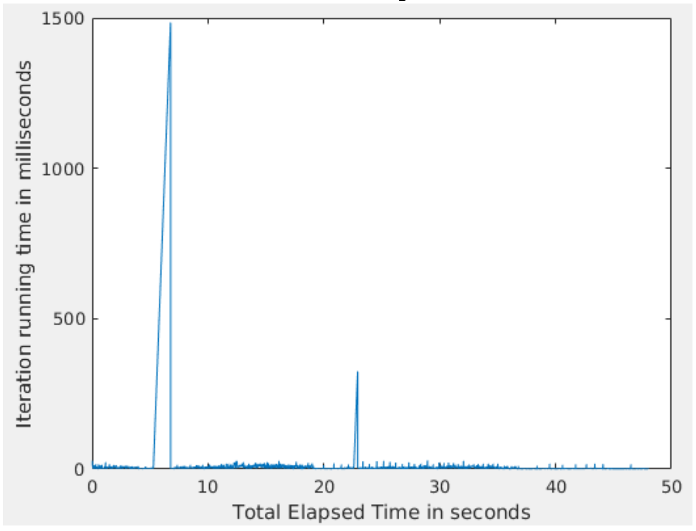

Figure 3.9: Iteration execution time over iteration number

3.4 Greedy Algorithm

Figure 3.10 shows the result of the algorithm for the Roombots size of 1.2.

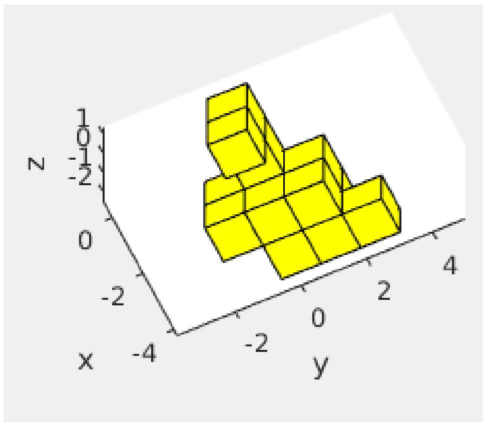

Figure 3.10: Greedy Algorithm Result

This structure is made of 7 Roombots and was built in 0.295 seconds. Its metric value is 0.39 which is luckily close to the optimal solution.

3.5 Direct Voxelization with Verification

Figure 3.11 shows the result of this approach for a voxel resolution of 2.3:

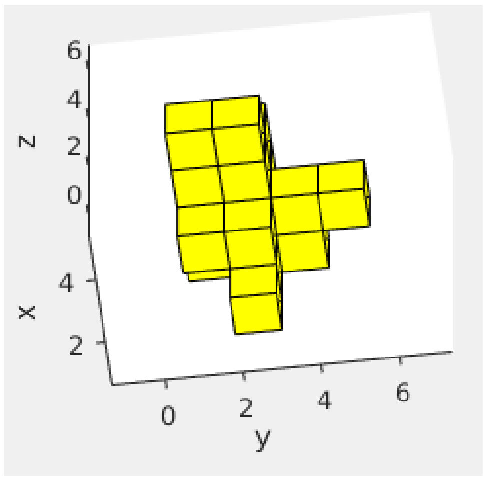

Figure 3.11: Direct Voxelization with Verification Result

This structure is made of 18 voxels or 9 Roombots and was built in 0.46 seconds.

To better illustrate the power of this method when it comes to performance and quality of results, we can increase the resolution to 0.7 for figure 3.12 and 0.4 for figure 3.13.

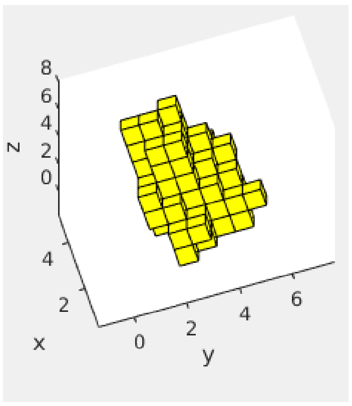

Figure 3.12: Direct Voxelization result with resolution 0.7

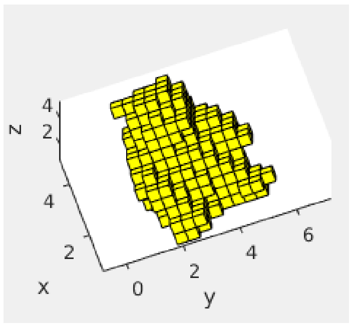

Figure 3.13: Direct Voxelization result with resolution 0.4

The two structures are made of 43 and 220 Roombots modules respectively and were built in 0.49 and 1.088 seconds respectively.

Of course, these Roombots modules counts are high, but given the nature of the object, it is especially hard to reconstruct it with lower resolutions.

3.6 More samples

For the end of this chapter, we have a look at two more objects through all of the steps of our project. One is another scanned real-world object, the other is a 3D mesh made directly on the computer.

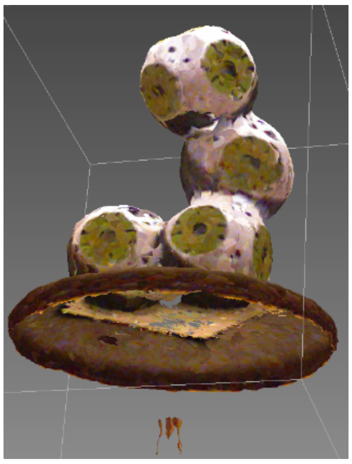

Figure 3.14: Roombots modules object scanned

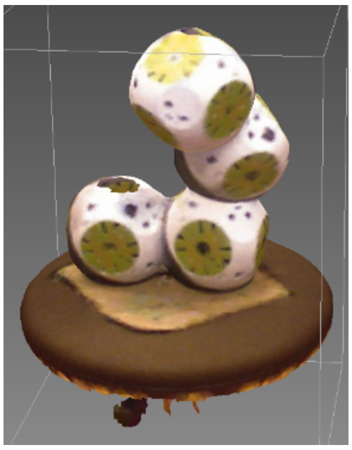

Figure 3.15: Finalized scan of the Roombots Modules

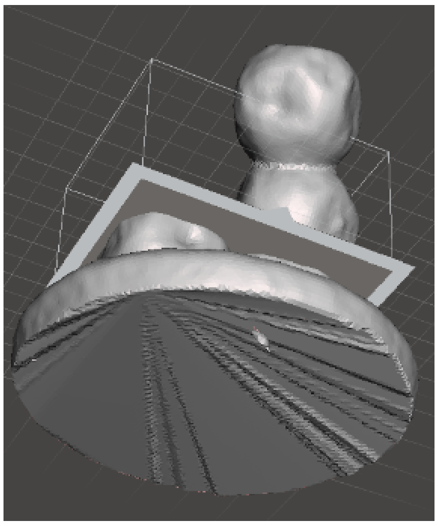

Figure 3.16: Roombots mesh with filled holes

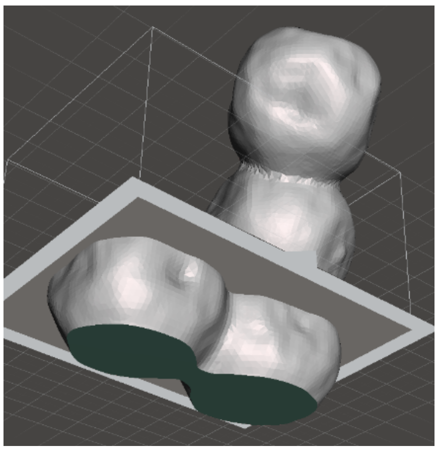

Figure 3.17: Roombots mesh with removed support

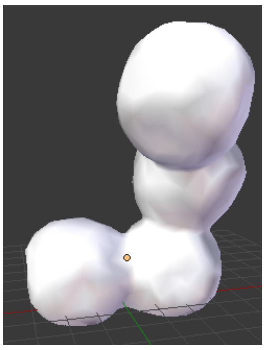

Figure 3.18: Final Roombots mesh with less vertices. This mesh is made of 1200 faces and 1200 vertices

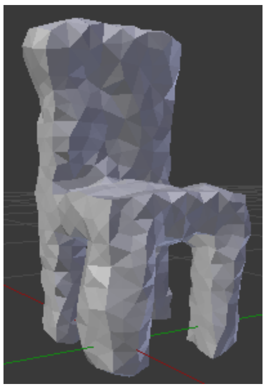

Figure 3.19: Chair 3D model with 2355 faces and 1158 vertices

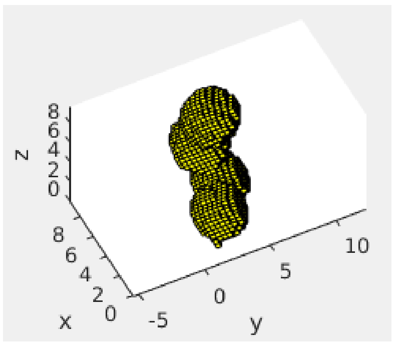

Figure 3.20: Roombots mesh voxelized in 1.716 sec. with 2330 voxels

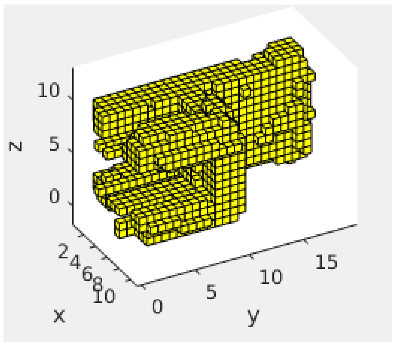

Figure 3.21: Chair mesh voxelized in 1.883 sec. with 2132 voxels

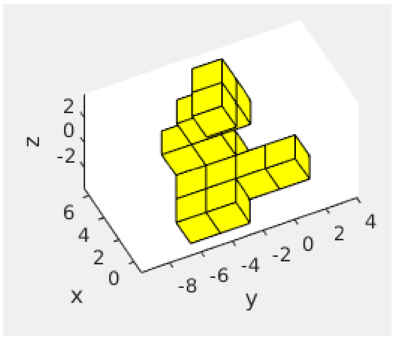

Figure 3.22: BFS algorithm result for the Roombots mesh with 7 Roombots, a resolution of 1.9, a metric value of 0.33 and obtained in 64.608 seconds

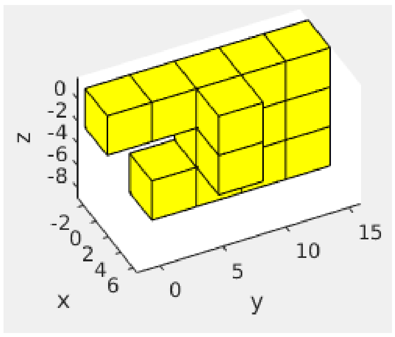

Figure 3.23: BFS algorithm result for the chair mesh with 7 Roombots, a resolution of 3.57, a metric value of 0.35 and obtained in 2.637 seconds

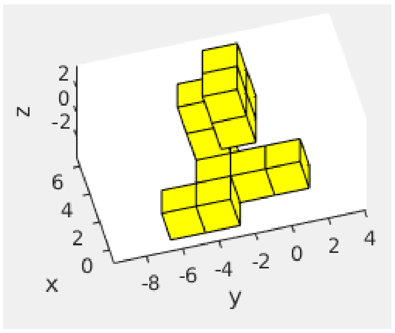

Figure 3.24: Greedy algorithm result for the Roombots mesh with 6 Roombots, a resolution of 1.9, a metric value of 0.32 and obtained in 0.384 seconds.

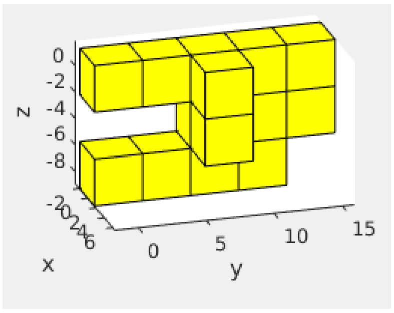

Figure 3.25: Greedy algorithm result for the chair mesh with 7 Roombots, a resolution of 3.57, a metric value of 0.32 and obtained in 0.345 seconds.

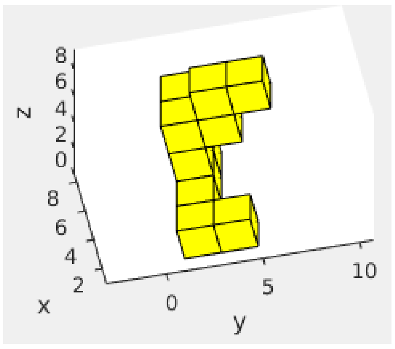

Figure 3.26: Direct voxelization with verification result for the Roombots mesh with 6 Roombots, a resolution of 1.9 and obtained in 0.25 seconds

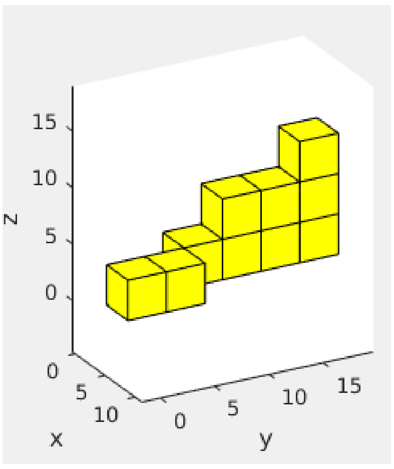

Figure 3.27: Direct voxelization with verification result for the chair mesh with 5 Roombots, a resolution of 3.57 and obtained in 0.381 seconds

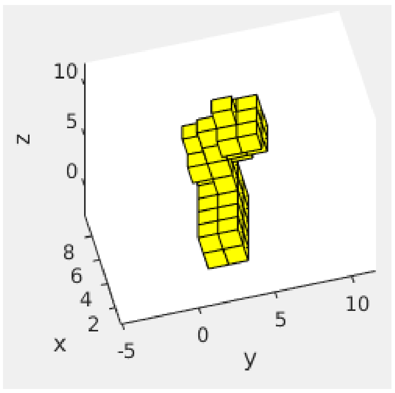

Figure 3.28: Direct voxelization with verification result for the Roombots mesh with 23 Roombots, a resolution of 1.3 and obtained in 0.257 seconds

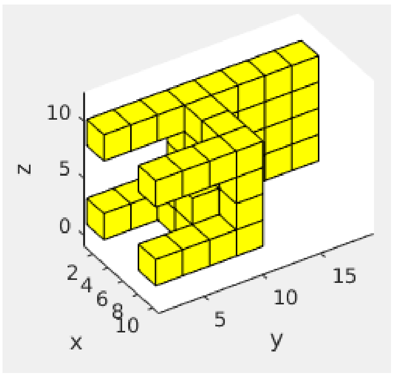

Figure 3.29: Direct voxelization with verification result for the chair mesh with 23 Roombots, a resolution of 2.3 and obtained in 0.559 seconds

Chapter 4

Discussion and Future Work

4.1 Scanning the object

3D scanning has a lot of room to improvement. Since the goal here is to make a user friendly approach, the scanning part of the project doesn't exactly come close to that.

One thing that could be done is to integrate the scanning part of the project, instead of using another software. The problem here, is that we would still need to correct the scan and remove unwanted parts such as the support. This could be solved by defining a special way of separating the interesting object from it's surroundings, for example, by making the Play-Doh form of a certain colour, and taking only this part of the mesh.

Another great thing would be to scan the interesting part of the scene in real time which would allow the user to modify the structure on the fly as they see the result of their construction at the same time.

Some of these problems however are out of scope of this project and are more of general problem and improvement possibilities to 3D scanning than the particular software of this project.

Another problem is that using our set-up, we were unable to capture any objects made of Play-Doh. These objects were too small for the scanner to properly align scans when rotating the object. To illustrate this problem, figure 4.1 shows a Sofa made with Play-Doh that we were trying to make, while figure 4.2 illustrates the aligning problems that we encountered. One solution would be to use more Play-Doh, or look for better software or hardware for 3D object scanning.

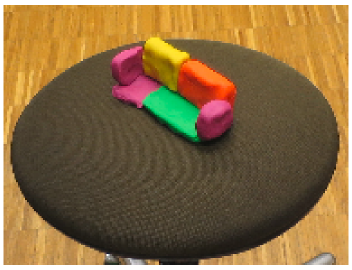
Figure 4.1: Real world Play-Doh object

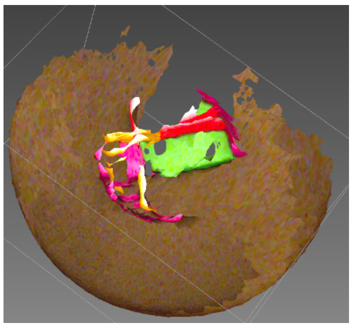
Figure 4.2: Aligning problems during the scanning of the Play-Doh object

4.2 Mesh Voxelization

The voxelization of a mesh is something that works reasonably fast and gives good results visually.

One thing that can be improved is adding the possibility of moving the grid of voxels over the original mesh. This would modify the voxelization and could in some cases improve the reconstruction quality.

The performance of the algorithm can also be improved. Instead of going through all the grid, the program could start by going near the locations of vertices of original mesh for example.

4.3 Performance of Algorithms and possibilities of expansion

The two algorithms (DFS and BFS) give results that are visually very pleasing. The solution they give is optimal as defined by the value metric function.

Looking at the time however, we realize that the algorithms are too slow for practical uses. Since the goal is to make something user-friendly, these algorithms will not work in real time.

The graph of figure 3.8 shows us very well the problem. The exponential increase of created children at a given depth is what causes the enormous performance loss. This loss is of course amplified at the next level, as per definition of BFS algorithm The DFS approach is the same when it comes to performance results. That approach could potentially lead to better performance if we knew in advance what is the optimal solution, but this is not the case here, and it would only do so when lucky.

The graph of the figure 3.9 confirms however that the algorithms are as fast as they can be. Each iteration of the algorithm is very fast, and the time does not increase in any particular manner. The largest spikes that are visible on the figure are caused by the operating system scheduling something else than the software at that particular moment most likely. In other words, the only room for improvement is to decrease the number of children opened. This can be possible if we decide to open the children more restrictively, but this way we run into the risk of decreasing the quality of the final structure and not going over all the possibilities.

For future development however, these algorithms can be augmented with different approaches that can lead to better performance. For example, the use of heuristics would be useful and may be possible for expansion of the search tree to decrease the running time by not opening nodes that are expected to provide worse results that those that are already opened. This is something that has been attempted to be added in the project but we couldn't find a heuristic to direct the search.

Finally, another part that can be improved is to automatically place the first Roombots module instead of asking the user to give this location manually.

4.4 Greedy algorithm

Greedy algorithm's results are exactly as expected when it comes to performance and quality. It us unlikely that we can see the optimal solution with it, but the greedy solution gives us one solution very quickly with better results that if we were to traverse a random branch of the tree. In a way, it shows us the maximum kind of speed-up we can have with heuristic algorithms, although with the worst kind of quality we could get from these algorithms.

4.5 Direct Voxelization with Verification

Direct voxelization gives very promising results for this problem. This method gives results of very good quality extremely fast when compared BFS or DFS, especially since we can have very high resolutions for which the two algorithms wouldn't terminate in any reasonable time. One thing that can be extended is to include voxels that are only partially inside the mesh while the centre point is outside of it. This could potentially lead to better results in some cases.

As of now, the verification of constructibility is far from optimal. The removal of cubes in the manner discussed can cause quality problems but also separate whole regions of the structure if a cube was removed while it was the only connecting point between to parts of the structure.

The biggest room for improvement here is then the verification part. This could be done in a more structured way sacrificing some of the great performance for better results.

User interaction here can also be an option. The program could signal problematic regions and ask the user to choose the best solution for himself. This approach would require the development of a user interface as well.

Another student's project this semester, made by Valentin Nigolian, is working on visualization and manipulation of Roombots structures using virtual reality in an Oculus Rift. His method could take the output of this project and modify a structure if needed.

Chapter 5

Conclusions

As the result of this project, we have a software that lets a user construct a Roombots structure from a scanned 3D object such as Play-Doh, but also an arbitrary 3D model. This has been achieved through the use of existing scanning software, and a custom process to organise Roombots modules into a structure covering the object.

The methods used are Artificial Intelligence search methods: Breadth-First-Search, Depth-First-Search and Greedy algorithms, as well as a method consisting in direct voxelization of the input mesh and verification of constructibility.

These methods allow us to obtain quality results, defined in the case of Artificial Intelligence search methods with a metric function.

While the direct voxelization method may not return an optimal solution due to its nature, it performs reasonably fast compared to Breadth-First-Search and Depth-First-Search and gives better results than the Greedy algorithm in the general case.

Unfortunately, Breadth-First-Search and Depth-First-Search algorithms are unable to obtain the optimal solution for structures that are too big in reasonable time because of their exponential growth in time. The direct voxelization results are not optimal. The verification of constructibility might cause structures to loose some of it's original visual quality be trying to remove voxels that could not form a Roombots module with another voxel.

The work in this project is a good basis for problems related to filling a mesh with voxelized structures such as Roombots. Time performance problems has been identified and the methods can be extended with a strong heuristic search method for Artificial Intelligence search methods and a strong verification of constructibility for direct voxelization method.

Bibliography

[1] Ayberk Özgür, Stéphane Bonardi, Massimo Vespignani, Rico Möckel, Auke J. Ijspeert. *Natural User Interface for Roombots.* The 23rd IEEE International Symposium on Robot and Human Interactive Communication 2014

[2] Romain Testuz, Yuliy Schwartzburg and Mark Pauly. *Automatic Generation of Constructable Brick Sculptures.* Eurographics 2013

[3] Microsoft Kinect. `https://dev.windows.com/en-us/kinect`

[4] KScan3D. `http://www.kscan3d.com/`

[5] MeshMixer. `http://www.meshmixer.com/`

[6] Voxelizer. `https://www.techhouse.org/~dmorris/voxelizer/`

[7] Möller-Trumbore intersection algorithm. `https://en.wikipedia.org/wiki/M%C3%B6ller%E2%80%93Trumbore_intersection_algorithm`