Institutionen för systemteknik
Department of Electrical Engineering

Final Thesis
Detection of Primitive Shapes in a Voxel Grid
by
Johan Serebrink

LiTH-ISY-EX--15/4903--SE
2016-02-12
<table>
<thead>
<tr>
<th>Presentation Date</th>
<th>Department and Division</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015-11-13</td>
<td>Department of Electrical Engineering (ISY)</td>
</tr>
<tr>
<td>Publishing Date (Electronic Version)</td>
<td></td>
</tr>
<tr>
<td>2016-02-12</td>
<td></td>
</tr>
<tr>
<td>Language</td>
<td>Type of Publication</td>
</tr>
<tr>
<td>English</td>
<td>Degree Thesis</td>
</tr>
<tr>
<td>Number of Pages</td>
<td>ISBN (Licentiate thesis)</td>
</tr>
<tr>
<td>41</td>
<td>ISRN</td>
</tr>
<tr>
<td></td>
<td>LiTH-ISY-EX—15/4903--SE</td>
</tr>
<tr>
<td></td>
<td>Title of series (Licentiate thesis)</td>
</tr>
<tr>
<td></td>
<td>Series number/ISSN (Licentiate thesis)</td>
</tr>
<tr>
<td>URL, Electronic Version</td>
<td></td>
</tr>
<tr>
<td>Publicaiton Title</td>
<td>Detection of Primitive Shapes in a Voxel Grid</td>
</tr>
<tr>
<td>Author</td>
<td>Johan Serebrink</td>
</tr>
<tr>
<td>Abstract</td>
<td>In 3D-optimisation of today a decrease in the amount of surfaces to render is often handled by replacing a neighbourhood of surfaces with a simplified and similar set of surfaces. In some cases when the surface structure is complex and highly symmetrical, the neighbourhood simplification might produce anomalies that break the symmetry in a visually detrimental way. By knowing the underlying primitive shape of the 3D-structure, simplified reproductions of a surface structure can be produced without risking the symmetrical integrity of the object. This report details the possibility of detecting the primitive shapes in a 3D-structure of voxels by analysing the available data. This is done by looking at neighbourhoods of voxels and using the gathered information to estimate normals and curvature data that indicates a primitive shape. Through categorisation, clustering and filtering of the data, primitive shapes can be detected with a degree of certainty. Utilising a few different approaches an analysis is performed on the efficiency in the detection of primitives on the surface of voxel structures. The report concludes that with a sequence of easy-to-follow steps reliable detection of primitives can be performed.</td>
</tr>
<tr>
<td>Keywords</td>
<td>Voxel Grid, Voxels, Analysis, Primitive Shapes, Primitives, Surfaces, Curvatures, Normals, Clustering.</td>
</tr>
</tbody>
</table>
Final Thesis

Detection of Primitive Shapes in a Voxel grid

by

Johan Serebrink

LiTH-ISY-EX--15/4903--SE
2016-02-12

Supervisors: Gustaf Johansson (Donya Labs AB)
Jens Ogniewski (Linköping University)

Examiner: Ingemar Ragnemalm (Linköping University)
Abstract

In 3D-optimisation of today a decrease in the amount of surfaces to render is often handled by replacing a neighbourhood of surfaces with a simplified and similar set of surfaces. In some cases when the surface structure is complex and highly symmetrical, the neighbourhood simplification might produce anomalies that break the symmetry in a visually detrimental way. By knowing the underlying primitive shape of the 3D-structure, simplified reproductions of a surface structure can be produced without risking the symmetrical integrity of the object.

This report details the possibility of detecting the primitive shapes in a 3D-structure of voxels by analysing the available data. This is done by looking at neighbourhoods of voxels and using the gathered information to estimate normals and curvature data that indicates a primitive shape. Through categorisation, clustering and filtering of the data, primitive shapes can be detected with a degree of certainty.

Utilising a few different approaches an analysis is performed on the efficiency in the detection of primitives on the surface of voxel structures. The report concludes that with a sequence of easy-to-follow steps reliable detection of primitives can be performed.
1 Introduction

Computers can effortlessly do complex tasks that for humans require strenuous thinking. Mathematics that leave students struggling can be done thousands, if not millions, of times a second in even the simplest of circuits. With this ease of calculating, it might be seen as surprising that a computer, while so capable, is left overwhelmed doing tasks that humans do all the time subconsciously, like seeing a cat and instinctively knowing that it is not a dog.

While difficult, the problems are being solved step by step. A key issue in many of these situations is the sheer number of cases that need to be handled and steps have to be taken to simplify the problem, or find smart solutions, to allow computers to complete the task in reasonable time.

In this thesis the problem presented is not to detect as complex shapes as cats or dogs but to find some of the most primitive shapes there are, specifically spheres, cylinders and planar surfaces. To do this, a discretisation of the primitive is performed by converting it to a structure in a 3D-grid of voxels, called a voxel grid.

1.1 About voxels

While voxels as a concept is still unfamiliar to most people, its use has become gradually more widespread over the recent years. When asked to describe a voxel it is easy to fall back on the origin of the word itself which is a blend of the words ‘volume’ and ‘pixel’, a ‘volumetric-pixel’ or perhaps 3D-pixel. Similar to the 2D-image produced by a digital camera, made up of pixels, a voxel grid represents a 3D-image of an object, made up of voxels.

![Figure 1. A composition of filled voxels, together forming a structure. An individual voxel has been marked in grey. (CC BY-SA 2.5 Vossman)](https://example.com/image)

Just like the image of a camera the resolution, how many elements per distance unit there are, has vital importance to the quality of the representation. Voxel grids have resolution expanding in three dimensions, contrasting to only two dimensions for ordinary images. This
quickly results in a very large number of voxels with increased resolution. To keep down the complexity and memory allocation of the voxel grid each voxel is binary, it is either filled or unfilled, and cannot adopt a scale of colours.

1.2 Purpose and Aim
In 3D-model simplification a fundamental goal is to reduce the number of surfaces in a model without affecting the shape and appearance significantly. One way to do this is by reduction, where a neighbourhood of surfaces is replaced with a similar set of surfaces that contain fewer triangles or quads. Sometimes when trying to simplify a highly symmetric structure, like a cube, the reduction might sometimes cut a corner or create a corner that does not have 90-degree angles which skews the symmetry.

These symmetric disruptions can easily be avoided by generating a primitive shape from scratch. The information about the shape however is not always available, or might never have existed in the first place. If we would be able to reverse engineer the parameters of the shape then it might be possible to do a more reliable 3D-model simplification.

This kind of reverse engineering aimed at extracting shape parameters has been done in images for a long time, sometimes to detect ellipses and lines [1]. Some of the same principles that were employed in detection of arrangements in images can be translated to voxel grids.

The aim of this thesis is to test whether primitive shapes (alternatively called primitives) can be detected in a voxel grid and if so, how much various factors affect the result. What methods could be used and how do these differ from each other.

The result of the work is an analysis method and implementation that can take a binary voxel grid containing voxel structures and make an estimation of a few types of primitives within the grid. In order to produce test cases, software to generate voxel structures will also be procured. To render data and results an implementation in Unity 5 was developed and used.

1.3 Problem Formulation
- What different ways are there to calculate normals on voxels in a voxel grid?
- How does the resolution of the voxel grid affect the ability to detect shapes?
- What is a good procedure to detect primitive shapes?
- What kind of shapes can be reliably detected?
- What kind of shapes cause problems for the algorithm?

1.4 Scope and Limitations
The focus of this thesis is how to detect primitive shapes. As there is a time limit of 20 weeks, the scope of the project is limited in some areas.

1. Integration
As implementations using voxels are becoming more common so are the ways to represent them. The project is not aimed at fitting existing programs or to create data that is compatible with existing software. Data representations are specifically built to satisfy functionality within the project, not outside it.

2. Depth of Analysis
There is a virtually unlimited amount of information that can be gathered about a voxel structure in a voxel grid. It would be unrealistic and a waste of resources to try to gather as much information as possible. Therefore information of most importance to the project is prioritised and some common types of information is neglected, as the necessity and usefulness of extracting them is not outweighing the resources, mainly time, to be spent. This includes, but is not limited to, distance fields.

3. **Primitives to Detect**

   It is not realistic within the scope of the project to detect all kinds of primitives. Three distinct shapes were prioritised; spherical, cylindrical and planar surfaces.

1.5 **Donya Labs AB**

The thesis project was done at Donya Labs AB in Linköping. Donya Labs is one of the leading companies in automatic 3D-optimisation and the developers of the software Simplygon.
2 Theory

The usage of voxels as a way to depict 3D-stuctures is rather novel. However, in recent years the usage of the technique has been expanded into a wide variety of fields. The gaming industry has adopted the use of this technique for simulating dynamically modifiable worlds and structures [2] [3] [4]. Within the field of medicine voxels have been used to represent scanned data from imaging techniques [5]. Steps have also been taken to using voxels for motion planning in robotic applications [6].

Voxel grids are ordered and member voxels can be easily accessed which is a key feature in utilisation. The efficient access allows for speedy analysis of structures which other representations, such as the point cloud representation, lack. It is however possible to see a voxel structure as a simplification of a point cloud structure and as such many methods of analysis can be deducted from those used with point clouds. The two representations have a lot in common and therefore both will be described in the theory section of this report.

2.1 Point Clouds

A point cloud is a set of data points in a coordinate system, in this report specifically a set (cloud) of 3D-coordinates (points) in a 3D-space. The points are not inherently ordered but may together form a shape, very much like an ordinary cloud which, being made up from huge amount of small droplets, form structures that may be interpreted as figures.

![Figure 2. A large green point cloud in 3D-space complemented by a few smaller point clouds of different colours.](image)

A single point cannot tell us much about a structure and there must be a limit where there simply are too few points to be able to tell what a structure represents. A part of this thesis approaches this subject because there needs to be a certain amount of points, or voxels, to reliably detect shapes.

2.2 Voxels and Voxel Grids

In this report a voxel is either filled or unfilled. The voxels are stored in a voxel grid and can be seen as a binary 3D-array. Voxel can be graphically represented as points or cubes. Cubes make for a good visualisation of the structure but require significantly more processing power than points. This is further explained later.

2.2.1 Voxel Neighbours and Neighbourhoods

A single voxel on its own holds very little information about the structure it belongs to. As such it is necessary to group voxels together to learn more about the structure to which they are a part. The set of voxels in an area close to a certain voxel is called a neighbourhood of
voxels. Nearby voxels to another are called neighbours. Since voxels need neighbours to form structures different types of neighbourhoods have been classified as follows.

**Class 1 Neighbours**
The class 1 neighbours of a voxel are the ones reachable within one step in the voxel grid while not allowing diagonal movement. If thinking of a voxel as a cube with each side to a neighbour this neighbourhood is a class 1 neighbourhood. A voxel has 6 class 1 neighbours.

![Figure 3. The class 1 neighbourhood voxels of an unfilled voxel. The voxels have been made smaller to allow a better view.](image)

**Class 2 Neighbours**
Consider a completely filled 3x3x3 sized voxel grid, to the center-voxel the remaining voxels apart from the 8 corner voxels are the class 2 neighbours. A voxel has 18 class 2 neighbours. This neighbourhood class has not been used in this report.

**Class 3 Neighbours**
The class 3 neighbours of a voxel are the ones reachable within one step in the voxel grid while allowing diagonal movement. Consider a completely filled 3x3x3 sized voxel grid, to the center-voxel the remaining voxels are the class 3 neighbours. A voxel has 26 class 3 neighbours.
2.2.2 Different Types of Voxels

Depending on the position and neighbours of a voxel it is possible to group it into different types. The following denominations are used extensively throughout this report.

*Hidden Voxel*

A hidden voxel is a filled voxel where all class 1 neighbours are filled. When representing voxels as cubes, the hidden voxel cannot be seen from outside the structure.

*Surface Voxel*

A surface voxel is filled, not hidden, and has at least one of its class 1 neighbours unfilled.

*Border Voxel*

The voxels that are next to out-of-bounds of the voxel grid are called border voxels. In a 3x3x3 sized voxel grid all but the center voxel are considered border voxels.

2.2.3 Different Types of Neighbourhoods

There are different ways to form neighbourhoods around voxels. In this report two ways of procuring a neighbourhood around a voxel have been tested. The first is one referred to as the *distance method* and the second as the *flood fill method*. The methods usually filter the gathered voxels, for instance only allowing surface voxels that are not border voxels.

1. **Distance Neighbourhood**
   
   The distance method gathers all voxels within a certain distance of a pivot voxel.

2. **Flood Fill Neighbourhood**

   The flood fill method gathers all voxels doing a breadth first search to a certain depth on the class 3 neighbours from a pivot voxel.

The main difference between the two methods is that the distance method allows segments of unconnected voxels to be considered a neighbourhood, as opposed to the flood fill method where there must exist a connected path between all neighbourhood members.
2.3 Voxel Grids as a Subset of Point Clouds
Many of the characteristics of binary voxels are shared with the points of point clouds. The main difference is that voxels are evenly spaced in a grid while points in point clouds can be anywhere in space. This makes voxels easier to work with. The voxel grid allows checking neighbours quickly while also reducing the storage footprint [7].

2.4 Estimating Normals and Curvatures
The estimation of normals and subsequently curvature has been essential to the reliable detection of primitives. Without good estimates the initial guesses for primitives lose precision, leading to false positives or extensive error correction during analysis. In this report three methods of normal estimation are presented.

Curvature detection is based on the preceding normal estimation. Using a neighbourhood of surface voxels and their corresponding normals, principal directions and principal curvatures are estimated for all surface voxels. More on this later in the report.

2.4.1 Normal Estimation Methods
A normal is a vector defining the tangent plane of a surface point, in the case of a voxel grid this point is represented by a pivot surface voxel.

A primitive normal is a normal estimated from a class 1 neighbourhood, which point toward the unfilled voxels. A primitive normal can only take one of 26 different directions but are cheap to compute. Only surface voxels have primitive normals, voxels without any filled class 1 neighbours have no primitive normal.

The three methods for estimating normals that will be explored in this report, names chosen to reflect their procedure, are as following.

- **Basic Normal Estimation**
  This method is performed by estimating a primitive normal for each neighbour in a flood filled surface voxel neighbourhood and estimating an average from these primitive normals.

- **Inverted Filled Space Normal Attraction**
  This method estimates a center point of all distance neighbourhood voxels around a pivot voxel, pointing a vector from the pivot voxel to the center point, negating the given vector and normalizing it.

- **Planar Fitting of Flood Filled Surface Voxel Neighbourhood**
  By using a flood filled surface voxel neighbourhood a planar fitting is done using total least squares and the normal is set as the tangent vector of this plane, this closely resembles that of Hoppe et al [8] [9]. The tangent direction is corrected according to the primitive normal of the pivot.

More is written about the effects of the different methods in chapters 4.2.2, 4.2.3, and 4.2.4. The implementation is covered in chapter 5.3.1.

2.4.2 Curvature Estimation Method
Using a flood fill surface voxel neighbourhood around a pivot voxel allows for a reliable estimate of principal curvature values and principal directions. The approach of Zhang X. et
al [2] only requires neighbourhood voxels and their normals to do this, not requiring mesh reconstruction or surface fitting and this is the method used here.

2.4.3 Definition of Curvature
Curvature is a measure of how bent a surface is at a certain point. It correlates to the radius as follows:

\[ \kappa = \frac{1}{R} \]

Where \( \kappa \) is the curvature and \( R \) is the radius of the osculating circle [10].

Principal curvature has two curvature values, \( \kappa_1 \) and \( \kappa_2 \). They denominate the minimum and maximum curvatures of a surface, at a certain point [11]. In this thesis they are crucial to the detection of spherical and cylindrical surfaces. The directions of the principal curvatures are called the principal directions and they are perpendicular to one another. The maximum and minimum curvature of a surface align with the principal directions. A spherical surface should have equal \( \kappa_1 \) and \( \kappa_2 \), curving just as much in any direction, but not zero as this would indicate a planar surface. A cylindrical surface should have one principal curvature value equal to zero and the other reflecting the radius of the cylinder, since the curvature along the length of the cylinder, being straight, has an infinite radius. A planar surface should have both principal directions at zero, having infinite radius of the osculating circle in any direction. Testing for these properties allows for clustering of surface voxels with similar values, together forming clusters that define primitives in the voxel grid.

2.4.4 Determination of Radius According to Curvature
The radius is the reciprocal of the curvature. The two values \( \kappa_1 \) and \( \kappa_2 \) can be converted to radius of the respective osculating circle. When mentioned in the report the radii is denominated \( R_1 \) and \( R_2 \). They are however sorted so that \( R_1 \) is the osculating circle radius of the principal curvature closest to 0 and the other way around for \( R_2 \). That means the \( R_1 \) radius represents the shortest radius and \( R_2 \) is the longest.

2.5 Primitive shapes
In this report a primitive is defined from a surface voxel cluster of a voxel structure. Voxel structures representing primitives contain filled voxels and shells of primitive surfaces are not supported. That implies it is not possible to find a primitive structure inside another, unless that primitive causes an empty space. If a primitive is completely enveloped in another it cannot be detected.

2.6 Representation of Results
The results produced by the analysis of the voxel grid is represented as clusters of surface voxels with similar properties like curvature and general normal direction. A cluster of voxels is marked as either a spherical, cylindrical or planar surface.
3 Related Works

There are several ways to analyse a 3D structure to find incorporated primitive shapes. Considering that the voxel grid is quite niche as a representation of 3D objects not much has been written specifically on the subject of analysis of voxel structures. There are however works analysing point clouds that uses a voxel representation as an intermediate step in the detection sequence [7] [8] [9]. As point clouds and voxel grids are relatively similar structures, methods for normal and curvature estimation designed for point clouds can be used on voxel grids as well.

The primary focus of this chapter is previous works on normal and curvature estimation. Provided normal and curvature data of surface voxels the final stages of the detection is grouping surface voxels with similar properties together which in this thesis follows no particular preceding works.

3.1 Estimation of Normals and Tangent Planes

In the work of Hoppe et al. an algorithm for reconstructing surfaces from point clouds is presented [8]. The algorithm estimates a tangent plane in each point. By gathering a neighbourhood around a point a tangent plane for the point is estimated by the least squares method. Employing a covariance matrix and extracting the eigenvalues produces the normal that defines the tangent plane. The normal is not consistently oriented and the paper further describes the problem with tangent plane orientation and proposes a method to produce a consistent tangent plane orientation. The same approach in estimating a normal for a point is used by Yoo in his paper on surface reconstruction [9].

Using the same method is possible in voxel grids by procuring a neighbourhood of only surface voxels. The problem with the orientation of the normal is not as difficult to handle in a voxel grid as it is in a point cloud. In the voxel grid the structures are filled and this makes it easy to check if the normal of the tangent plane need to be turned around.

3.2 Estimation of Curvature

A way to estimate curvature in point clouds is presented in the work of Zhang et al. [12]. In this paper on curvature estimation they present an interesting method requiring only the neighbourhood points around a point and the normals of the neighbours. They call the algorithm Chord And Normal vectors (CAN) method.

The CAN method estimates the principal curvature of a chosen point. This is done by estimating curvature between the chosen point and its neighbours respectively. These curvature values are then put into a value function and by using least squares fitting on the values it is possible to estimate the principal curvature values and the principal directions.

The work is novel in the way that it does not need to perform mesh reconstruction or surface fitting when estimating curvature, suiting the voxel grid representation well.
4 Algorithm Choices

There is not one universal way to detect primitive shapes. The same way that there is not a single perfect way to design a solution to the detection problem. A few choices have to be made somewhere as to what shapes to prioritise and how many calculations that are to be executed until considered done. In this section of the report focus lies with solution design choices and their effect on the result.

4.1 Choice of Primitives

The focus of this work has been to detect primitives. The primary choice of primitives to detect was made by Donya Labs as a part of the specification of the thesis work. They wanted to know more about the possibilities of using voxels as a part of their simplification process, mainly on highly symmetrical structures, or structures with underlying primitive shapes. The three primary primitives that were chosen were spheres, cylinders and planar surfaces.

An example of this would be the typical city block building with protruding balconies. A person could easily tell if the primary structure of the building could be represented as a rectangular cuboid. By converting the building to a voxel structure in a voxel grid, could the underlying primitive structure be detected? A building consists mostly of planar surfaces and the decision to detect planar surfaces stems from this example.

The spheres and cylinders are very different to the planar surfaces in that their curvature has to be taken into account during detection. A cylinder can also be considered a combination of an ellipse of one infinite radius cropped by two planar surfaces necessitating the combinatorial detection of a curved surface in combination with a planar one.

4.2 Assigning Normals to Voxels

As explained earlier, three different methods of normal estimation have been tested. They all have different strengths and weaknesses which will be further explained here.

4.2.1 Difficulties in Normal Estimation

A couple of phenomenon become apparent when determining normals. One is in this report called edge-deviation, caused by the neighbourhood extending over an edge, resulting in deviation of the calculated normal. The deviation becomes more prominent the closer to the edge the neighbourhood pivot voxel is and gives rise to areas close to edges with normals that are not properly representing the surface they belong to. How far from the edge an area is affected is closely related to the depth of the neighbourhood. The deeper the neighbourhood extends the more likely it is to become affected by an edge in the structure.
Borders of the voxel grid can also give rise to poor normals, the occurrence here called **border-deviation**. Consider a simple voxel structure partially out-of-bounds of the voxel grid. When estimating the normals of surface voxels close to the border, the neighbourhood available can get skewed to one side (the other side being out-of-bounds and non-existent). This can result in undefined behaviour leading to poor normal estimations.

### 4.2.2 Basic Normal Estimation (BASIC)

As indicated by the name the main benefit of the basic normal estimation method is simplicity in the implementation. The requirements are a surface voxel neighbourhood and predetermined primitive normals of all members of the neighbourhood. The primitive normals can be calculated just using class 1 neighbours as explained in chapter 2.4.1.

The sum of all the neighbourhood members’ primitive normals is normalised and used as output. The expectation is that the larger the neighbourhood is, the less each crude primitive normal will matter and the final average will be a good estimate of the normal of the surface. However the larger the neighbourhood is the more likely it is to hit an anomaly in the voxel structure, causing edge-deviation and resulting in poor estimates. This phenomenon is more apparent in this method than in the others since it requires a particularly large number of surface voxel neighbours for an accurate normal estimate.

### 4.2.3 Inverted Filled Space Normal Attraction (IFSNA)

This method uses the fact that voxel structures are solid (not hollow) in the voxel grid. All filled neighbours around a pivot voxel are gathered by using the distance method. By gathering all the positions of the neighbourhood voxels, a center point can be calculated. It is then possible to point a vector from the pivot voxel to the center point, negate the vector to make it point outwards from the structure and then normalise it, resulting in a reasonably accurate normal.

The method uses the fact that distance neighbourhoods usually do not extend far enough to include multiple surfaces. And that even when this happens it is likely that the closest surface takes precedence since more voxels belong to it, only resulting in a slight deviation. It cannot be dismissed completely that the normal is turned into the structure, though this is very rare,
and can be checked by comparing the estimated normal with the primitive normal to make sure of correct orientation. The method is affected by edge-deviation but is not as dependent of a large neighbourhood as the basic normal estimation method.

4.2.4 Planar Fitting of Flood Filled Surface Voxel Neighbourhood (PFFFSVN)
This method is highly influenced by previous works described in chapter 3.1. By estimating the tangent plane of a surface voxel neighbourhood, the normal is perpendicular to that plane. In this method a planar fitting is done on a flood filled surface voxel neighbourhood. By doing a least squares planar fitting on the neighbourhood, an estimate of the tangent plane can be computed and subsequently a normal [13].

Planar fitting can result in a normal that is turned into the structure, depending on the tangent plane seed. By comparing to a primitive normal the correct direction of the estimated normal can be ensured. This method is not affected by unconnected areas like the inverted filled space normal attraction method, and not as reliant on large neighbourhoods as the basic method. This method was the primary choice of the final primitive detection in this thesis.

4.3 Estimating Curvature
Curvature estimation is reliant on the normal estimation performed previously. Using the work of Zhang et al [12] reliable principal curvature values and directions can be acquired from just surface voxel neighbourhoods and their normals, given that the normals are properly estimated. Since normals are affected by edge- and border-deviation, so are in turn the curvature estimations.

A lot of papers focus on curvature estimation in 3D-meshes. It is possible to convert the surface voxels and their estimated normals into meshes and then run one of these methods on the result. This is a valid and proven method, but it adds another step in the processing resulting in a more complex solution with risk of larger error propagation. Therefore the choice was made to use the method of Zhang et al [12].

4.4 Software Development
A large part of the project consisted in developing the software required to detect primitives in voxel grids. However not all of the development work was put into the analysis software. To run the analysis there was a need for specified voxel grids to feed the analyser as well as rendering software to actually see what was done.

4.4.1 Rendering Software
It might be questioned whether or not rendering was necessary to complete the project. The detection of primitives could be measured in statistics, tables and textual output, the analysis of the voxel grid is done just as well without actually seeing the result rendered. Stepping aside from the question whether or not rendering is necessary and instead focusing on the effects of rendering, it can be said that rendering has many benefits but comes with its own set of obstacles to overcome.

There are two primary ways to render voxels, as dots or as cubes. Dots are computationally cheap to render but lacks depth and cannot be viewed up close with good result since they are always represented by one pixel on the screen. Cubes on the other hand produces great result both close and viewed from afar. They are much more computationally expensive to render though, and requires sophisticated methods to produce good results in real-time. The solution
was to render large amounts of voxels as dots and small amounts as cubes depending on the
current voxel grid being rendered. The cut-off was set to a constant measured against the
number of filled voxels to be rendered.

The main benefit of rendering the voxel grid is that we can get a view of the structures
generated. This allows the viewer to quickly see if the generated voxel structures are as
expected. This becomes especially apparent when working with rotated structures. It also
allows for the use of randomly generated structures when testing the algorithms presented in
this thesis. Without rendering it would not be as simple to see the correlation between the
structure and the result of the detection of primitive structures. Allowing for the use of
generic structures can highlight flaws in the given parameters and algorithms. There is also
the possibility that multiple randomised structures can intersect in interesting ways that would
otherwise have gone unnoticed without the ability to render the voxel grid.
5 Implementation Design

During the course of the project three important pieces of software were developed. The first was a voxel grid generator, allowing the production of suitable test cases to take place. The second was a renderer with which voxel grids, results and data could be displayed. The third and foremost was the analyser were voxel grids could be loaded, analysed and results produced.

5.1 Generator

The generator was designed to produce voxel grids containing test cases to evaluate the analyser. The generator was designed to run command line input or scripts with command line inputs. The generator supports different sizes of voxel grids and can generate primitives at given positions with specific parameters. There is also support for exporting the normals of the primitives generated, allowing for comparisons between estimated normals and the correct ones of each voxel.

![Figure 6. The used voxel grid generator software.](image)

5.2 Renderer

The renderer was designed to handle four different kinds of data. The types are point grids, point clouds, line grids, and line clouds. All types are supported with colours per element allowing for separation between different areas in a voxel grid.

- **Point Grids and Point Clouds**
  
  A point grid is a representation of a voxel grid, being a set of points aligned as a three dimensional Cartesian grid. It is a simplification of the point cloud, which is essentially the same thing without the inherent structure of a grid. The point grid uses significantly less storage space, not having to store the positions of the points, just requiring the size of the grid and whether or not a point is filled or unfilled (true or false). A point cloud can be used to display cylinder-axis point estimations or sphere center estimations to mention a few.

- **Line Grids and Line Clouds**
A line grid is a way to represent lines with starting points originating from positions in a Cartesian grid. A line grid can be used to display normals of voxels, all originating from a point in a grid – a voxel. The line cloud is a representation of lines without structure, allowing lines to start and end anywhere at the expense of storage space used. A line cloud can be used to display principal curvature directions or cylinder-axis.

5.3 Analyser
The analyser is by far the most extensive part of the implementation and also the software actually detecting surface clusters adhering to a primitive. The detection process is a collaboration between the detection of spherical, cylindrical and planar surfaces. By combining the results of the three, the best detection can be selected reducing the amount of false positives in the result.

The analysis starts by estimating normals and curvature data for the surface voxels of the voxel grid. This information will be shared by all the following detection algorithms trying to cluster spherical, cylindrical and planar surfaces.

5.3.1 Estimating Normals
Three different methods were tested when estimating normals for the surface voxels of the voxel grid. In the following chapters, if not specified otherwise, the normal estimation method of use is the planar fitting of flood filled surface voxel neighbourhood method, the motivation behind this can be read about in chapter 4.2.4.

*Basic normal estimation (BASIC)*
By using a large amount of primitive normals and calculating the average, it is possible to make an accurate estimate of the normal for that surface voxel.

The method starts with a surface voxel and its flood fill surface voxel neighbourhood. All the neighbours have their respective primitive normals estimated, which are then summed up and normalised. The result is the basic normal estimate.

*Inverted filled space normal attraction (IFSNA)*
This method estimates a normal using the center point of the neighbourhood. Normals are only estimated on surface voxels. Knowing that voxel structures are filled and usually quite large compared to the size of a neighbourhood, it is possible to estimate the normal of a surface voxel from the ‘mass’ of the nearby structure.

The method starts with the pivot surface voxel to estimate a normal for and its distance neighbourhood. By summing all the neighbourhood voxel positions and normalising the result, it is possible to obtain the center point of the neighbourhood. Drawing a vector to this center point from the pivot voxel, normalising and negating (‘inverting’ the direction) a decent normal estimate is achieved.

*Planar fitting of flood filled surface voxel neighbourhood (PFFFSVN)*
It is possible to estimate a normal of a surface voxel if the tangent plane of that voxel is known. Knowing the flood fill surface voxel neighbourhood around the pivot voxel a total least squares planar fitting is performed on the neighbours to estimate a normal [13] [9]. This method has a flaw in that sometimes the normal is turned pointing into the structure instead
of out of it [9]. By testing if the normal’s dot product against the primitive normal is positive the direction can be easily ensured. The resulting normal is the one used.

5.3.2 Estimating Curvature
The curvature estimation is done according to X. Zhang et al [12]. Needed for the estimation are the flood fill surface voxel neighbourhood around the pivot surface voxel and the estimated normals of the neighbours and the pivot. The normals have been estimated earlier.

This method estimates curvature between the pivot voxel and the neighbours one by one. Using a least squares approach and a standardised eigenvector solver the eigenvalues corresponding to the principal curvatures are extracted for the pivot.

5.3.3 Clustering Planar Surfaces
The normals of the surface voxels are vital in the detection of coherent planar surfaces. As mentioned previously the normal estimation is not perfect and suffer from edge-deviation and border-deviation which needs to be handled by the algorithm. To work around this deficit, the planar surface detection algorithm performs a sequence of different steps of which some are iterative to refine the planar surface clusters. The steps are as follows:

1. Find primary planar surface clusters.
2. Refine planar surface clusters.
3. Expand planar surface clusters.
4. Trim redundant clusters.

**Find primary planar surface clusters**
A set of markers for all surface voxels is created, initialising all members as ‘not-added-to-cluster’. The algorithm then iterates all surface voxels, creating a new cluster when it finds a voxel that is ‘not-added-to-cluster’, this voxel is called the pivot.

When creating a new cluster a flood fill is started from the pivot voxel. The flood fill expands on the class 3 neighbourhood and adds any voxel marked ‘not-added-to-cluster’ as long as it satisfies an angle-deviation constraint from the pivot voxels’ normal to its own, these surface voxels are added to the cluster and marked ‘added-to-cluster’. The flood fill does not expand on any neighbour that is added to a cluster. Eventually the flood fill stops and a cluster is formed and added to the set of clusters.

This will cause all surface voxels to belong to a cluster. The results can look like shown in figure 7.
Figure 7. An example of primary surface clusters. Depicts the normal vectors coloured according to cluster on the corner of a cube structure viewed from inside.

Often the result is not well aligned with the planar surfaces of the voxel grid due to corner pieces being picked first, among other reasons. Some of the clusters are significantly larger than others, hinting at likely planar surfaces.

**Refine planar surface clusters**

To refine the clusters that were found previously, the order of the clusters are first sorted in regard to the member count of the cluster, placing the largest clusters first and the smallest last.

For each cluster, an average normal is calculated, the members are then compared with this average normal to find the member whose normal has the smallest angle difference to this average. This member becomes the pivot voxel and is stored separately. This is repeated for each cluster, after which the clusters are discarded.

Starting with the new pivot voxels, new clusters are produced with a surface voxel flood fill, within the allowed angle deviation, expanding on the class 3 neighbourhoods. If a pivot voxel has been already added to a cluster it is discarded. This behaviour will favour planar surfaces with many members, which will be expanded first and expanded so that the best representative is used as a reference normal. A second pass is performed, creating clusters from the surface voxels that were not added to any cluster after having gone through all pivot voxels. This step can be iterated for even better clustering. In figure 8 an example of this refinement is displayed.
Expand planar surface clusters
As long as there are planar surfaces that have a noticeable number of voxels unaffected by edge-deviation and border-deviation, a consistent cluster should exist in the middle of the surface. In this step we attempt to expand these clusters so that they acquire more members.

A cluster already has its average normal and center of mass-point calculated. These can form the definition of a plane. Using this definition of the plane an expansive flood fill along the plane over surface voxels is performed from all the members of the cluster. If a new surface voxel is within a certain distance of the plane it is added to the cluster and removed from its previous cluster. A surface voxel can only be re-assigned to a new cluster once in this step, otherwise the members of the larger clusters could be stolen by smaller clusters handled later in the sequence.

The purpose of this step is to handle the edge-deviation and border-deviation. This step is not affected by the directions of the normals but rather the positions of the surface voxels, basically removing the issue with poor normals close to edges and borders. When this step is done the result can look something like depicted in figure 9.
The algorithm relies on a later step to remove the redundant parts of the clusters. This is done by making an estimation of which primitive, regardless of type, the voxel most likely belongs to.

**Trim redundant clusters**

After the previous steps have been completed there will be a number of really small clusters that clearly are too small to adequately represent a surface. A trim is performed that removes all clusters under a certain member count threshold. This threshold can be set freely, it is however possible to say that clusters with less than 3 members should always be trimmed, as planar surfaces cannot be defined in those cases. Deciding to trim clusters with a higher member count than 2 is subject to the risk of removing small yet correct planar surface clusters. Taking into account that the expanded planar surface clusters from before often have high member counts this risk can be worth taking as it might decrease the amount of false planar surface clusters significantly.

### 5.3.4 Clustering Cylindrical Surfaces

This step focuses on detecting the side surface of a cylinder, as opposed to the base- and top-surfaces of a cylinder. The base and top is clustered and detected as planar surfaces, see chapter 5.3.3.

A cylinder can be distinguished on the great difference between the two principal radii in a surface element. According to T.-T. Tran [14] a difference factor of over 100 indicates a cylinder in a point cloud. In this algorithm a lower factor, 50, has been used to make up for the less dense point data of the voxel grid. Theoretically the difference ratio should be infinite (the larger of the two radii being infinite), but the detection of principal radii is not flawless and therefore this threshold is required. It is possible that a torus become clustered as a cylinder following these criteria and it is not dealt with in this thesis.
The clustering is performed in four steps.

1. Find cylindrical surfaces.
2. Cluster cylindrical surfaces.
3. Extend cylindrical surface clusters.
4. Remove redundant clusters.

Each step improves on the previous resulting in an increasingly detailed clustering.

**Find cylindrical surfaces**

An iteration is performed over all surface voxels, each surface voxel has its principal curvature analysed. Converting the principal curvatures to the radii of the osculating circle the following test is performed. If \( R_1 \) (the shortest radius) is a factor of 50 smaller than \( R_2 \) (the largest radius) then the surface voxel is considered cylindrical. All the surface voxels that satisfy the criteria are marked 'cylindrical' for later processing.

**Cluster cylindrical surfaces**

Improving on the previous step, the surface voxels marked cylindrical are clustered. Iterating all surface voxels marked cylindrical a flood fill is performed expanding on the class 3 surface voxel neighbourhood. Only the cylindrical-marked voxels are expanded on.

As a new, unclustered, cylindrical-marked surface voxel is found a new cluster is created. The first voxel becomes the pivot voxel and it is added to the cluster. The pivot voxel’s principal curvatures are converted to osculating-circle-radii, \( R_1 \) and \( R_2 \). The shortest radius, \( R_1 \), is the radius of the cylinder. This is used as a reference during the following flood fill.

The flood fill starts with the pivot voxel and compares the reference radius to the radius of the neighbours. It will only test neighbours that are cylindrical-marked previously. The neighbour radius must be within a certain accepted-deviation-factor. The factor was set to a value of 0.3 meaning that the radius could deviate 30% from the pivot before it would be considered a different cluster. The result of this step can look like figure 10.
Figure 10. A cylindrical clustering of surface voxels marked cylindrical. The surface voxels are viewed from inside the structure after having removed all hidden voxels during rendering.

It is clearly noticeable that not all surface voxels become marked as cylindrical, even if they should be. A lower threshold factor when finding cylindrical surfaces will increase the amount of available cylindrical-marked voxels. This will however also increase the amount of surface voxels that have principal curvatures poorly reflecting the actual radius of the underlying cylindrical structure as well as increase the amount of false positives appearing on planar surfaces.

**Extend cylindrical surface clusters**

Using the clusters acquired in the previous step an extension is done according to the average radius of the cluster and an estimate of the cylinder-axis. The average radius, \( r_{average} \), is the average \( R_1 \) radius of all included members.

Every member of the cluster is a surface voxel and all surface voxels have estimated normals. In combination with knowing the average radius of the cylinder an estimate of a point on the cylinder-axis can be computed for each member. Estimation of a cylinder-axis point is performed as follows:

\[
\bar{p}_{cylinder-axis,i} = (-\bar{n}_i) \times R_{average}
\]

Where \( \bar{n}_i \) denotes the normalised normal vector of member \( i \), \( R_{average} \) is the scalar average radius of the cylinder-cluster. This is performed for all members \( i \) belonging to the cluster.

A linear total least squares fitting is performed according to D. Eberly [13] using the estimated cylinder-axis points which results in a direction-vector pointing along the length of the cylinder. A center of mass calculation on all the points provides an origin of the cylinder-
axis direction vector. Having acquired this information it is trivial to check for distance to the linear cylinder-axis from voxels in the voxel grid [15]. This will be used when extending the cylinder to new surface voxels.

Extending the cluster is necessary in order to negate the effects of border- and edge-deviation. The extension is executed similarly to that of the planar surface clustering, a surface voxel flood fill expanding on the class 3 neighbourhood. The position of the voxels to be added must be within a certain distance to the cylinder-axis close to the average radius of the cluster calculated earlier. The threshold must allow for some error in the estimation of the cylinder-axis but cannot be too large either, removing surface voxels from other cylinders. The threshold was set to ±1.5 units of the average radius.

The expansion is performed by starting with the largest cluster then moving down the hierarchy. It is worth noting surface voxels can be members of many different clusters at the same time. This overloading of surface voxels can be easily corrected later by prioritising large clusters over smaller ones. Since not all surface voxels of the underlying cylindrical structure were part of an initial cluster prior to this step, this expansion is likely to be flawed. The cylinder axis might be skewed and the radius might not be sufficiently accurate. Nevertheless the expanded cluster is often significantly better than the previous one and can be iterated for even better result.

Iteration of this step poses a few new obstacles to overcome. Since the clusters now contain members affected by border- and edge-deviation the normals are not reliable when re-estimating a cylinder-axis. A new average radius can be estimated from the previous cylinder-axis to the member voxels. Using the old cylinder-axis, normals can be re-estimated by just normalising the vector from the closest point on the cylinder-axis to the voxel concerned. These values can be expanded upon again, allowing for iterative refinement of the cylinder-axis and radius estimate. It is also possible to remove members of the cluster that do not fit the new cylinder-parameters.

This iteration continues until no change in member count is detected or when the member count change becomes negative for any reason, limiting the cluster expansion to additive changes in member count. The result of the iterative expansion can look like figure 11:
Almost all surface voxels become clustered correctly. Around the edges of cylinders small clusters shine through that will be removed in a later stage. On the planar surfaces a few false positive clusters have been added. These will likely be removed when combining the clusters since detected planar surfaces will be much larger than any one false positive cluster.

**Remove redundant clusters**

When the iterative process is done redundant and overlapping clusters should be removed. Any cluster under a certain threshold can be removed under the assumption that if a cluster is too small, it is likely not correct due to the small amount of members to perform an estimate on.

Overlapping clusters can be removed by iterating all clusters with the largest in member-count first in falling order. Large clusters are likely correct to a higher degree and should take precedence over smaller. Allowing every surface voxel to only belong to a single cluster will result in surface voxel clusters that are large and usually spanning entire cylinders without interruption by smaller clusters.

### 5.3.5 Clustering Spherical Surfaces

A group of spherical surface voxels should have about the same $R_1$ and $R_2$ since the principal curvatures are about the same. This attribute is exploited in the spherical clustering. The clustering is performed in four steps resembling that of cylindrical clustering:

1. Find spherical surfaces.
2. Cluster spherical surfaces.
3. Extend spherical surface clusters.
4. Remove redundant clusters

**Find spherical surfaces**
A test is performed on all surface voxels to see if they likely are part of a sphere. A surface voxel is considered spherical if $R_1$ and $R_2$ have the same sign and are within a threshold of each other. The difference threshold between $R_1$ and $R_2$ was set to 10% of the length of $R_1$. A higher percentage will increase the number of spherical surfaces detected but also the number of false positives. False positives tend to appear around the edges of cylinders and on planar surfaces. This is to be expected, since principal curvature naturally has the same properties as spheres in these places. A planar surface can have two principal radii that are large but close to each other. A cylinder affected by edge-deviation can have a set of surface voxels close to the base or top that are curved as much toward the edge as around the side, also fulfilling the spherical properties.

By assigning a strict difference threshold, the amount of false positives can be kept low. By reducing the amount of false positives these voxels will be less likely to form connected clusters with many members. This is desired since member count is used later as a way to determine whether a cluster is actually spherical or not.

**Cluster spherical surfaces**
The surface voxels that in the previous step became marked ‘spherical’ are iterated, all markers being initialised as ‘unvisited’. When an unvisited voxel is detected a surface voxel flood fill is started with the newly found voxel as pivot. The flood fill expands on the class 3 surface voxel neighbourhood on voxels that are marked spherical and unvisited.

The difference between the average radius of the pivot and the expanded voxel is tested. If the absolute value of the difference is under a percentage of the pivot’s $r_{average}$, it passes. In the testing 20% of $r_{average}$ of the pivot was used as threshold. The threshold has little effect on the outcome unless it is too low, splitting a sphere cluster into many where only one should exist. If set too high there is a small risk that a cluster from one sphere structure spreads to another that has a different radius. This risk can usually be overlooked since edge-deviation acts as a barrier that separates a spherical structure from another.

**Extend spherical surface clusters**
When finding spherical surface voxels, not all the voxels that should belong to the sphere get detected, partially because of edge-deviation caused by intersecting structures but also because the spherical voxel detection is applied to a surface of limited resolution. To make up for these missed voxels an extension of existing clusters is performed.

A different approach has been tested on extending spheres. Instead of extending a plane outward in a direction as was done with the planar and cylindrical surfaces a sphere is fitted to the cluster member voxels. The fitting is done with least squares according to D. Eberly [13]. This method iterates to come up with the best fitting center point from which an average radius can be calculated to all members of the cluster. When an estimate of the sphere has been made, voxels close to it can be added to the cluster, resulting in a more complete sphere. If a surface voxel was within 1.3 units from the estimated radius of the sphere, it was added to the cluster. This threshold can be chosen arbitrarily, a high value will remove voxels from other structures close to the sphere. A too low value will result in no new voxels being added.
to the previous sphere, ending the iteration. The iteration continues until no positive change can be detected in the member count from one iteration to another.

The iteration does not allow removal of surface voxels from the cluster, slightly limiting the potential of this method, but it also limits the number of iterations. Using this method, a cluster cannot iteratively deviate significantly from the original cluster.

![Figure 12. A spherical clustering after refinement. A few false positives can be seen on planar surfaces.](image)

**Remove redundant clusters**
All clusters with a member count under a given threshold is removed. The threshold can be chosen arbitrarily but the amount of redundant clusters are generally much lower when detecting spheres than planar or cylindrical surfaces.

**5.3.6 Combining Clusters**
When all different primitives have been detected it is possible to combine the separate results to a complete clustering. This can be done by sorting all clusters after size together, so that the largest spherical, cylindrical and planar clusters can be processed in relation to each other and not individually.

When this is done the clusters are iterated, largest first. The active cluster tries to claim any member voxel and can do so as long as no cluster before has claimed that voxel. If a voxel already has been claimed by a cluster it will be removed as a member. This behaviour will only allow a voxel to belong to one single cluster after the iteration has finished. As the largest clusters can claim voxels first they are more likely to claim voxels typically resulting in a few large clusters. An example of this merge is shown in figure 13.
Figure 13. A combined clustering. Inside the black square, clusters are marked according to cluster type, red clusters are spheres, green clusters are planar surfaces and blue are cylindrical clusters. Outside the black square, clusters have random colours.
6 Results and Evaluation

A summary of the results produced from the work performed over the course of the thesis work can be found here. Focus has been put on making the result approachable and some examples have been provided to complement the descriptions.

6.1 Normal and Curvature Estimations

Estimating normals proved generally successful albeit with a few surmountable limitations. The main parameter that profoundly changed the result of normal estimation was the depth of neighbourhoods. A shallow depth, moving just 2–4 voxels away, limited the effects of border- and edge-deviation, providing precise cuts between different surfaces.

A shallow depth has its own set of limitations too. Shallow depths sometimes result in planar surfaces splitting into different clusters. The larger the structure, the more dominant this behaviour becomes. This can cause problems when clustering planar surfaces and when estimating curvature.

An example of how the flood fill depth affects the deviation in the normals can be seen in Figure 14. Here the maximum deviation is measured, the estimated normals’ deviation from the actual normal using the planar fitting of flood filled surface voxel neighbourhood method.

![Figure 14. The results from measuring the maximum angle between actual normal and estimated normal on a planar voxels surface, tilted 5°, unaffected by border- and edge-deviation.](image)

In the case of the clustering of planar surfaces the effect can be negated by allowing larger angle deviation in the refining process. This is not without risk. If the angle deviation is large, curved surfaces and structure edges are more likely to be clustered together as a planar surface.

How the neighbourhood depth in the normal estimation affects subsequent curvature estimations can be difficult to assert. The difference from normal estimations is that there is another set of parameters to take into account. The curvature estimation in itself has a neighbourhood depth. This adds complexity since the curvatures are affected by both the
quality of the normals and the amount of normals used in the estimation. In addition to this, the curvature is affected by structure edges just like the normal estimation.

What that means is if a shallow depth is used in normal estimation it is possible to still make a good estimation of curvature by increasing the depth of the curvature estimation. This will make the curvature estimation more affected by structure edges though.

All this concludes in a balancing act. It is necessary to balance the depth of neighbourhoods with the allowed angle deviation to suit the structure that should be detected. There is no silver bullet combination using these algorithms that can detect any shape of arbitrary size. It is however possible to find a structure among other structures if the parameters are correctly set to reflect the situations that can appear.

6.1.1 Differences in normal estimation methods
There are a few issues that have to be taken into account when estimating a normal of a voxel. The first one is the neighbourhood. The neighbourhood of voxels in proximity to the pivot voxel must be accessed when estimating a normal. This is simply relating to the fact that a single voxel has no information about the surface inherently, it being binary.

Trying to estimate which voxels are beneficial and which are detrimental when estimating a normal is a complex problem. Often the voxels close by the pivot voxel belong to the same structure and should as such be beneficial to the estimation method. This approach to finding relevant neighbourhood voxels is the basis in the estimation of normals (and curvature) in this work.

Two different methods of acquiring neighbourhood voxels have been presented. The distance method is easy to implement and does not require a search. The flood fill method however guarantees that a neighbourhood is connected with filled voxels.

Generally speaking the flood fill method has higher potential than the distance method. The flood fill does not always expand in three dimensions like the distance method and can be more restrictive in what neighbours to visit e.g. when expanding only on surface voxels. Often a distance search performed for a surface voxel will visit unfilled voxels roughly half of the time, since usually the structure stretches further than the depth of the neighbourhood and the other half is unfilled space. This large number of unnecessary visits gets even worse when only surface voxel neighbours are desired as is the case when using the planar fitting of surface voxels described previously.

It is however possible to argue for the use of the distance method for small neighbourhood depths where the overhead of the flood fill method is a problem, even for extracting surface voxel neighbours, although it is more appealing in use with methods like the inverted filled space normal attraction where a large number of hidden voxels are beneficial.

In figure 15 the effects of different neighbourhood depths can be seen in relation to the size of the sphere and the method used for estimation. The plot indicates that for very small neighbourhood depths the basic method using primitive normals is actually preferable when detecting spheres. Another interesting conclusion is that the effect of increasing the neighbourhood depth quickly diminishes. Considering the amount of neighbours increase exponentially with the neighbourhood depth keeping the depth limited is essential to reducing execution time of the analysis.
Figure 15. The results from running three different methods of normal estimation on two differently sized spheres with varying neighbourhood depth. The average angular difference between the actual spherical normal and the estimated normal of all surface voxels is plotted on the y-axis. BASIC – basic normal estimation, IFSNA – inverted filled space normal attraction, PFFFSVN – planar fitting of flood filled surface voxel neighbourhood.

In figure 16 the same test has been repeated with a cube tilted 22.5 degrees around the x-axis and then 22.5 degrees around the y-axis. This tilting is done as to not favour the basic method. Here it can be seen that the basic method works best of the methods given a sufficiently small neighbourhood depth. A noticeable difference from the spherical test is the deterioration of the result with too large a neighbourhood depth. This is likely due to the edge-deviation that becomes dominant across the estimations. As previously anticipated, the edge-deviation is particularly noticeable on the normals of the small cube regardless of method.

The IFSNA (inverted filled space normal attraction) and PFFFSVN (planar fitting of flood filled surface voxel neighbourhood) methods follow each other’s result closely on both the cube-test and the sphere-test. On the cube-test however the results are highly dependable on the size of the cube, an inclination not noticed on the sphere. This would suggest that the estimation of normals is not as affected by the size of a sphere as it is by the size of a structure with edges. Particularly in regard to the neighbourhood depth where too deep is highly detrimental to the estimation of normals on a cube structure.
**Figure 16.** The results from running three different methods of normal estimation on two differently sized cubes with varying neighbourhood depth. The average angular difference between the actual normal and the estimated normal of all surface voxels is plotted on the y-axis. **BASIC** – Basic normal estimation, **IFSNA** – Inverted filled space normal attraction, **PFFFSVN** – Planar fitting of flood filled surface voxel neighbourhood.

### 6.2 Clustering Surfaces Representing Primitives

Having concluded that there is no perfect way of detecting primitives under arbitrary circumstances, how well do the algorithms hold up in acceptable conditions?

The best result was achieved when detecting spheres, this is likely due to the symmetry of the structure and the efficient least squares iterative process used directly on the voxels in the estimation of the primitive parameters. A sphere is also limited in size in a way that planar surfaces or cylindrical surfaces not inherently are. A sphere has just a certain radius, it will not extend infinitely in any direction. A cylinder, on the other hand, might be arbitrarily long, just as a planar surface can extend infinitely. This property makes them difficult to detect, especially if the surface is broken up by other surfaces. If it was known a cylinder existed as a structure in the voxel grid, the detection could be made more accurate, the same thing true with a planar surface.

On the subject of determining what a good procedure to detect primitives is, a reasonable assumption based on the results acquired here is that the less estimated data that has to be processed the less error propagation can occur. As such, an approach that works primarily on the voxels themselves is to be recommended, since then there would be no need to rely on estimated values. Normals and curvature data could be used to create seeds for likely positions of primitive structures, but that the final clustering is done primarily on voxels instead of estimated normals and curvature data.
6.2.1 How Resolution Affect the Detection of Primitives

When representing a primitive resolution has a significant effect on the ability to detect it. A high resolution however comes with its own set of obstacles. While border- and edge-deviation diminishes with increasing resolution the time required to process the voxel grid with all its neighbourhoods and clusters quickly grows.

As the resolution goes up, provided a well-balanced set of parameters, the detection gradually improves until eventually curvature becomes too low to detect within the neighbourhood depth and curved surfaces starts being considered planar. This is due to the neighbourhood becoming very small compared to the full structure following the reasoning given previously in chapter 6.1.1.

6.2.2 Weaknesses in Used Methods

A few types of shapes cause problems for the proposed detection methodology. As mentioned before, border- and edge-deviation causes some difficulties when detecting primitives, these shortcomings can be addressed by using shallow neighbourhoods or optimising allowed angles.

One of the issues in the detection is the dominance of large clusters in the refining steps. While accounting for the weighting between clusters, always prioritising the larger cluster is not always the best approach. Surfaces that might be better suited as a part of a smaller and more distinct surface can sometimes be consumed by a large cluster laying claim to voxels that ambiguously belong to both clusters according to the methods used. Whether this poses an issue mostly depends on the application utilising it though.

Other shapes than spheres, cylinders and planar surfaces are sure to cause problems for the detection process. For instance, the introduction of a cone as a part of the voxel grid is not handled by current procedures and will likely cause a mix of detections along the surface. This subject has not been tested as a part of the project. The decision to not test these shapes has mostly been a matter of resources available. A continued study of ellipsoids as a primitive would bridge an interesting gap that exist between primitive spheres and cylinders. More on this subject can be found in chapter 7.3.
7 Discussion
Detecting primitives in voxel grids has proven to be a very large subject to cover. A lot more can be done in the field. In this chapter a discussion on the used methods is held and suggestions are made on how to improve the results.

7.1 Detection Methodology
On the whole, the detection method used in this thesis works well in detecting the stated primitives. The detection of primitives that are not influenced by others performs particularly well. The issue with edge- and border-deviation remains. While this has been worked around perhaps there is a way to avoid the problems altogether. Possibly by working on the voxels themselves and by not using estimated normals as basis for the detection of coherent areas.

When improving on the present work, knowing the details are key to successful primitive detection. It would have been much easier to detect primitives with stricter demands on the voxel grid and the structures in it. For instance, disallowing border-voxels would completely remove the issue with border-deviation. This could be done by allocating a voxel grid after the voxel structures are defined so that no filled voxels exist on the borders of the grid.

More work can be done on learning the limitations of the proposed methods of detection, but also on posing the right questions. The question “How small can a planar surface be and still be detected?” is too ambiguous for the detection-problem. The detection is highly dependent on the presence of other structures. It is certainly possible to detect small surfaces, the problem occurs when the small surfaces are surrounded by other large surfaces with defects of their own. When posing the question these specifications should be made clear to improve a subsequent implementation process.

7.2 Uses for the Detection of Primitive Shapes
The clusters of coherent attributes corresponding to primitive structures in the voxel grid can have uses in different applications. The primary use intended when developing the methodology was to reverse-engineer voxel structures so that the attributes could be used for simplifying 3D-meshes, but possible uses extend past that.

By extracting the primitives of a voxel grid it could be possible to make a good estimate of what kind of object a voxel structure represents. Being able to catalogue voxel structures with these attributes has potential uses in computer-vision, where a computer might be able to tell one object from another easier.

There could also be uses within simple rendering of structures, where a voxel structure could be represented by a number of primitives just focusing on the major features and distinct formations, ignoring small or odd shapes.

7.3 Further Work
The detection of spheres is a subject that can be extended to include ellipsoids. An infinitely elongated ellipsoid could be seen as the surface of a cylinder. The detection of spheres and cylindrical surfaces could therefore be connected and could be an interesting continuation of the work.

The troubles with edge-deviation might be solvable through an iterative approach to the normal estimation. By not considering the normals constant after the initial estimation, but
instead subject to change according to the primitive shapes detected, it might be possible to iteratively improve the normals. This could have beneficial effects on primitive detection and the estimated normals at the same time.
8 References


